Host Integration Server 2009

Microsoft Host Integration Server

This version of Host Integration Server Help provides documentation and in-depth information about Host Integration Server 2009.

 Getting Started Information about new features, prerequisites, and topics for users who are new to Host Integration Server.

Programmer's Guide Information about how to develop Host Integration Server solutions for your environment.

 Samples Information about samples that are available for Host Integration Server.

 Community Resources Host Integration Server community resources such as blogs and forums.

Getting Started

This section contains information about new features, prerequisites, and topics for users who are new to Host Integration Server.

What's New for 2009

The following is an overview of the new features and enhancements in this release of Microsoft Host Integration Server and the BizTalk Adapters for Host Systems.

Expanded Microsoft platform support

- Windows Vista SP1, Windows Server 2008, Hyper-V
- Visual Studio 2008 SP1 and .NET Framework 3.5 SP1
- SQL Server 2008 and BizTalk Server 2009

Extended deployment options and management tools

- IPv6 networking for selected features
- Backup and recovery using VSS Writer
- New common Trace Viewer

Enhanced connectivity to host applications

- Generation of WCF Services based on Transaction Integrator objects
- Improved performance of Transaction Integrator and BizTalk adapter runtimes
- Dynamic Remote Environments for TI and BizTalk adapter
- New migration and deployment tools
- Design tool improvements, including COBOL/400 import/export
- Support for CICS v3.2 and IMS Connect v9

New WCF channel for WebSphere MQ

- With synchronous message correlation
- Built on system service model in .NET Framework 3.5
- Visual Studio project type to support optional data conversion
- Supports WebSphere MQ v7

Expanded connectivity to DB2

- Entity Designer and Entity Provider for DB2
- Improved performance when fetching data from DB2
- Support for DB2 for Linux and HP-UX platforms
- Compatibility with DB2 for z/OS v9 and DB2 UDB V9.5

Improved connectivity to host files

- Off-line data reader for host files ADO.NET provider and BizTalk adapter
- Visual Studio design tools for host files

BizTalk Adapters

This release of Microsoft Host Integration Server 2009 includes the following adapters:

The **Microsoft BizTalk Adapter for Host Applications** is designed for BizTalk Server. It is based on technology in Microsoft Transaction Integrator (TI) for Windows-Initiated Processing, which enables efficient client access to existing IBM mainframe zSeries (CICS and IMS) or midrange iSeries (RPG) server programs. The TI design tools are integrated with Visual Studio and BizTalk Server solutions, enabling IT developers to be highly productive when defining the client proxy and creating the XSD schema. For BizTalk Server administrators, the existing TI Manager, Microsoft Management Console (MMC) snap-in, has been enhanced to improve supportability and support the required remote BizTalk Server solution deployment scenarios.

The **Microsoft BizTalk Adapter for Host Files** is an advanced data adapter that enables IT organizations to access and integrate information stored in host file systems, including mainframe zSeries VSAM datasets and midrange iSeries physical files. The Visual Studio design tool is used within a BizTalk Server solution to define a metadata map of the host program-described files, which is then exported as XSD for use with the BizTalk adapter. The configuration wizards are integrated into the BizTalk Server administration tools, allowing IT pros to define dynamic send ports and static and solicit response receive ports, based on a simplified set of SQL commands (SELECT, INSERT, UPDATE, DELETE). This BizTalk adapter is based on a new Microsoft .NET Framework Data Provider for Host Files that maps SQL to non-relational host datasets and members, This makes it simple for non-programmers to read/write host file data sources.

The **Microsoft BizTalk Adapter for DB2** is a relational database adapter that allows IT professionals to access vital data stored in IBM DB2 database servers on remote host computing platforms, IBM mainframe zSeries and midrange iSeries (DB2/400), and also IBM DB2 Universal Database (UDB) on open platforms. Using standard SQL commands and a configuration wizard built into BizTalk Server administration tools, IT pros can create solutions that read and write to DB2 without any need for database programming. The new DB2 adapter, which is based an updated Microsoft .NET Framework Data Provider for DB2, supports a broad range of functions, including dynamic send ports, and static and solicit response receive ports.

The **Microsoft BizTalk Adapter for WebSphere MQ (Client-Based)** uses IBM WebSphere MQ Client (Base-Client) and IBM WebSphere MQ Extended Transactional Client (Extended-Client) APIs to communicate with remote MQSeries Queue Managers. The adapter enables BizTalk Server to communicate directly with MQSeries Queue Managers deployed on non-Windows operating systems, without needing to deploy and manage WebSphere MQ Server for Windows, to efficiently exchange messages with line-of-business applications across the enterprise. When used with the Base-Client, the adapter provides non-transactional message processing, guaranteeing only the delivery of messages. It is the responsibility of the application on the receiving end to handle any duplicate messages. When used with the Extended-Client, the adapter provides transactional message processing to guarantee once-and-only-once delivery of messages.

Transaction Integrator

Transaction Integrator (TI) is the synchronous COM+ or .NET Framework application integration solution in Host Integration Server. TI enables you to integrate mainframe-based transaction programs (TP) and AS/400 transactions with component-based Windows Server System applications when the following conditions are true:

- A synchronous or transactional solution is needed.
- Both the client and server systems are running at the time the call is made.

If you need an application integration solution that does not require the client and server systems to be running at the time the call is made, use an asynchronous messaging solution such as the MSMQ-MQSeries Bridge instead of TI. In an asynchronous solution, the middle-tier queuing system is running at the time the client issues a request message, the server retrieves the message and sends back the reply, and then the client receives the reply back from the middle tier.

With TI, you can integrate existing mainframe-based TPs with Windows-based COM, distributed COM (DCOM), or applications built on the .NET Framework. You might not even have to modify your TP if you have separated the business logic from the presentation logic. The wizards in TI guide you through the modification process, step by step.

With TI, you can preserve existing CICS and IMS TPs as you move to a three-tier client/server or Web-to-host computing environment. By using TI to invoke mainframe transactions, you can program in the visual object-oriented environments and programming languages that you know while you maintain access to host transactions.

TI supports both SNA connectivity and TCP/IP connectivity without requiring a host footprint or costly host transaction rewrites. You can choose SNA connectivity if you need two-phase commit (2PC), or choose TCP/IP connectivity if you need direct throughput. IBM has not implemented 2PC for the TCP/IP protocol, but for those cases where 2PC is not necessary, TCP/IP can give you direct connectivity.

True integration of online transaction processing (OLTP) with COM- or .NET-compliant systems means the integration of CICS and IMS with Windows-based solutions. CICS and IMS are widely used in the mainframe arena to create distributed OLTP solutions such as customer tracking and order entry. TI integrates CICS and IMS with COM by creating COM interfaces or .NET interfaces to the CICS and IMS transactions and then running the CICS and IMS transactions on the mainframe from Windows.

A TI component in a COM+ application works in concert with the TI run-time environment, Microsoft Distributed Transaction Coordinator (MS DTC), and the associated remote environment (RE) to drive a CICS or IMS TP. Together, they accomplish these tasks:

- Activate the host (mainframe) TP.
- Pass the parameters specified by the TI component to the TP.
- Run the TP.
- Return the results to the TI component.

When you deploy a TI component (a type library .tlb file) in a COM+ application, that COM+ application becomes a TI Automation server. When a client application invokes a method in that TI Automation server, Windows automatically starts the TI run-time environment in the associated remote environment to invoke the mainframe transaction that is associated with that TI method. Component Services in Windows 2000 automatically handles any class factory, early or late binding, or other internal operations needed. The invoked mainframe transaction can call other transactions on the mainframe before it returns the result to the COM-based client application through the TI Automation server.

Security Integration

Enterprise Single Sign-On (SSO) provides services to enable single sign-on for end users in enterprise application integration (EAI) solutions.

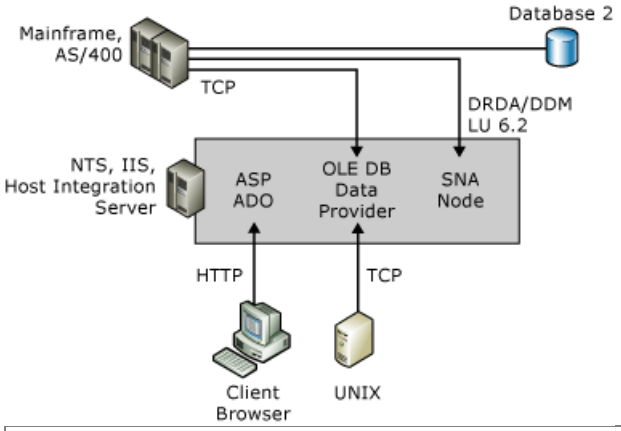
The SSO system maps Microsoft Windows accounts to back-end credentials. SSO simplifies the management of user IDs and passwords, both for users and administrators. It enables users to access back-end systems and applications by logging on only one time to the Windows network. For more information, see Security User's Guide.

Data Integration

Data services included with Host Integration Server 2009 enable you to interact with host data sources, including VSAM and DB2 systems.

The following diagram shows an overview of the Data Integration Services.

Data Integration Services



Note

If you are using TCP/IP to connect to the mainframe or AS/400, you do not have to install or configure the Network Integrati on Services. You can use the Application and Data Integration Services over TCP/IP.

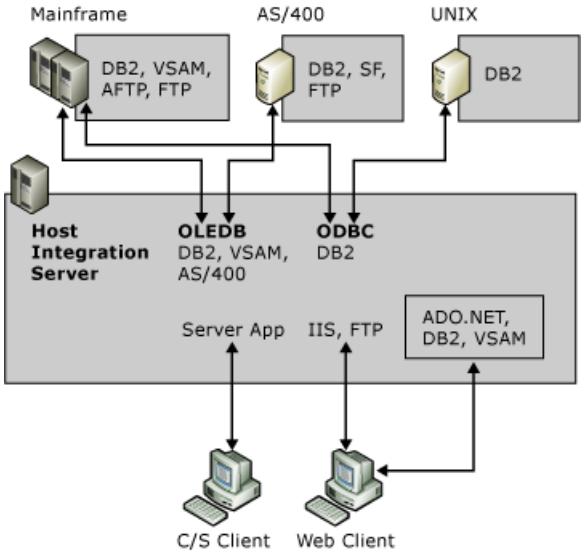
The following services are available.

- OLE DB Provider for AS/400 and VSAM
- OLE DB Provider for DB2
- ODBC Driver for DB2

OLE DB Provider for AS/400
The Data Access Tool simplifies configuration and access to AS/400 systems.

The following diagram shows an overview of the OLE DB Provider for AS/400.

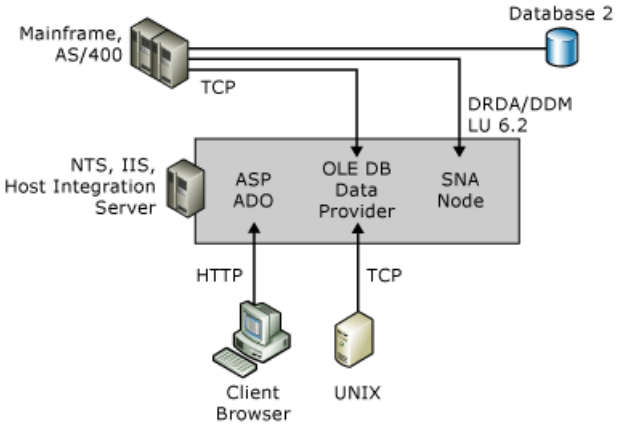
OLE DB Provider for AS/400



OLE DB Provider for VSAM
The Data Access Tool simplifies configuration and access to VSAM systems.

The following diagram shows an overview of the OLE DB Provider for VSAM.

OLE DB Provider for VSAM

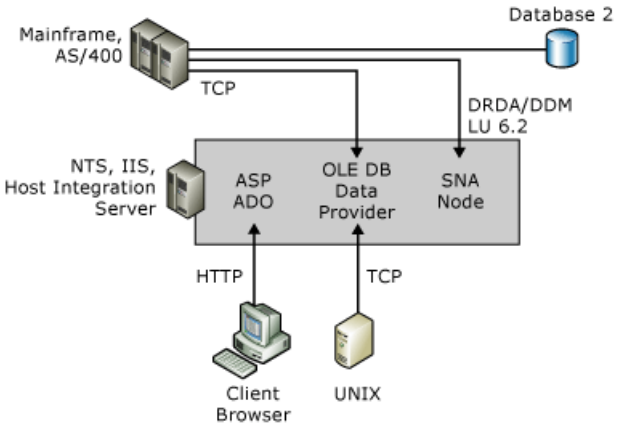


OLE DB Provider for DB2

The Data Access Tool simplifies configuration and access to DB2 systems.

The following diagram shows an overview of the OLE DB Provider for DB2.

OLE DB Provider for DB2



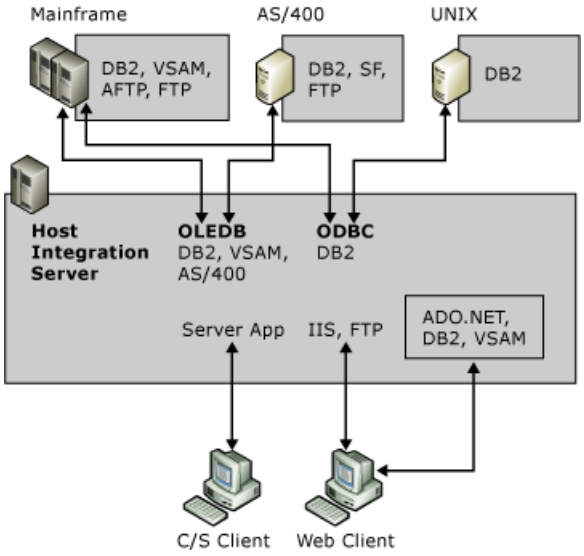
ODBC Provider for DB2

The Microsoft ODBC Driver for DB2 enables access over SNA LU 6.2 and TCP/IP networks to remote DB2 databases. This driver is implemented as an IBM Distributed Relational Database Architecture (DRDA) application requester that can connect to most DRDA-compliant DB2 systems, including MVS, VSE, VM, OS/400, AIX RS/6000, Sun Solaris, HP-UX, Digital/Compaq UNIX, OS/2, and Microsoft Windows 2000.

The driver can be used interactively or from an application program to issue SQL statements and execute DB2 stored procedures. Microsoft Office Excel users can import DB2 tables into worksheets and use Excel graphing tools to analyze the data. Microsoft Office Access users can import from and export to DB2. By using Microsoft Internet Information Server (IIS) on Windows 2000, developers can publish DB2-stored information to users through a Web browser interface.

The following diagram shows an overview of the ODBC Driver for DB2.

Diagram of ODBC Driver for DB2


Mainframe
AS/400
UNIX
DB2, VSAM, AFTP, FTP
DB2, SF, FTP
DB2
Host Integration Server
OLEDB
DB2, VSAM, AS/400
ODBC
DB2
Server App
IIS, FTP
ADO.NET, DB2, VSAM
C/S Client
Web Client

Network Integration

The IP-DLC link service is a Host Integration Server feature that provides SNA connectivity for applications that use dependent and independent sessions over a native IP network. It implements the HPR/IP protocol, which is also known as HPR over IP or Enterprise Extender. Each SNA packet is transmitted natively across the IP network as a UDP datagram.

This version of Host Integration Server supports the following physical connection methods to your host system:

- DLC 802.2 (for supported OSA adapters)
- SDLC
- Channel (ESCON and Bus & Tag)

Physical connections to a host system are provided with link services that support a variety of third-party adapters. A list of supported third-party adapters can be found in the online Help.

This version of Host Integration Server supports the following standard SNA protocols for communications:

- LU 1 and LU 3 (for host printing)
- LU 2 (for 3270 displays)
- LU 6.2 (for APPC)
- LU 0
- LUA

Note
If you are using TCP/IP to connect to the mainframe or AS/400, you do not have to install or configure the Network Integrati on Services. You can use the Application and Data Integration Services over TCP/IP.

Host Services
This version of Host Integration Server provides a range of services that extend mainframe services to LAN users. These include the TN3270 Service for mainframe environments, the TN5250 Service for AS/400 environments, and Host Print Service for both mainframe and AS/400 environments.

Administrative Services
This version of Host Integration Server gives network administrators tools and services to monitor and manage host resources in your enterprise. These services are available for both the hierarchical and peer environments.

Messaging

MSMQ-MQSeries Bridge enables your applications to exchange messages between IBM MQSeries and Microsoft Message Queue Server easily and efficiently. MSMQ-MQSeries Bridge provides connectionless, store-and-forward messaging across messaging systems and computing platforms throughout your network. To learn more about MSMQ-MQSeries Bridge, see Messaging User's Guide

Software Development Kit

This version of Host Integration Server includes a software development kit (SDK). This allows you to create custom applications, run samples, and use tutorials to learn more about Host Integration Server.

For more information, see Development.

Community Resources

Plug into the Host Integration Server community to connect with other developers and get answers to your questions, read the latest from bloggers, see webcasts, find out about events, and connect with MVPs.

Resource	Location
Host Integration Server blog	http://go.microsoft.com/fwlink/?LinkId=142377
Host Integration Server discussion groups	http://go.microsoft.com/fwlink/?LinkId=142378
Host Integration Server community on TechNet	http://go.microsoft.com/fwlink/?LinkId=142379

Planning and Architecture

This section summarizes the key points of how to plan a Host Integration Server deployment.

In This Section

Planning

Transaction Integrator Architecture

Planning

This section summarizes the key points of planning a Host Integration Server deployment.

In This Section

Planning Your Hardware

Planning 3270 Connectivity

Planning APPC Connectivity

Planning for Transaction Integrator

Planning Your Hardware

Because Host Integration Server can be used in a wide variety of situations, there is no simple formula for the amount of processing power a specific computer needs. The information provided in this topic can help you evaluate your hardware requirements, including:

- A list of elements that reduce or increase demands on hardware. This helps you estimate the general level of your processing requirements. For example, a Host Integration Server computer can serve as many as 1500 to 2500 users. This indicates a need for large amounts of processing power, perhaps a Pentium processor with 64 or 128 megabytes (MB) of RAM. However, if your servers each support 500 users and most of those users connect briefly and at random times during the day, less processing power is needed.

- Ways to evaluate the current demand on an existing Host Integration Server computer, and the ability of the existing hardware to support that demand. This gives you a valuable baseline from which to work.

In This Section

Variables Affecting Hardware

Many factors affect the amount of memory and processing power required by a particular Host Integration Server computer. The major variables are:

- Number of clients served.

  This is the most important variable affecting hardware requirements. If you support large numbers of client computers, you need not only servers that have ample memory and processing capacity, but also fast, powerful LAN adapters. Such LAN adapters include their own processor and memory, and can use direct memory access (DMA). Without such adapters, the network interface can become the bottleneck when many clients are being supported. Similarly, SNA adapters must be powerful enough to keep pace with the communications load.

- Demand generated by the clients (constant, intermittent, heavy, light).

- Number of Host Integration Server computers configured to work together within a particular subdomain.

  This is affected by the configuration of the Host Integration Server installation: centralized, branch, or distributed. Host Integration Server computers in a centralized or a distributed configuration can provide load balancing; they can each take a share of the connection load. Server computers in a branch configuration are separated by wide area network (WAN) links, and cannot readily provide load balancing because of the communication delays involved.

- Use of data encryption between clients and servers.

  Data encryption is not dependent on the number of clients being served but on the number of messages that are encrypted. As the number of transactions per second increases, the load on the CPU increases. Using 128-bit encryption instead of 40-bit encryption also increases the load on the CPU.

- Additional network services (file sharing, database access, mail service, and so on) provided by each server computer.

  It is difficult to predict exactly how much processing power is required to run multiple server applications. In many cases, your own requirements to meet a certain level of response time or transaction rate will indicate the specific hardware requirements for your environment.

See Also
Concepts
Estimating Hardware Demands
Best Practices

Estimating Hardware Demands

Use the information in the following table to estimate the amount of processing power required for your Host Integration Server computers.

Installation category	Hardware guidelines
Approximately 1500–5000 users for each server, with up to 15,000 sessions for each server	A multiprocessor system with at least 128 megabytes (MB) of RAM. Connections (such as DLC 802.2 or channel) that are fast enough to avoid bottlenecks (recommended). Multiple SNA adapters, except for channel. Multiple LAN adapters to avoid LAN bottlenecks. Optionally, dedicated Host Integration Server computers that do not provide file sharing or other LAN services (this depends on the load created by other LAN services).
Up to 600 users for each server (heavy use) or up to 1500 users for each server (light or intermittent use)	A Pentium computer, with at least 128 MB of RAM. Connections (such as DLC 802.2 or channel) that are fast enough to avoid bottlenecks (recommended). Multiple SNA adapters may be needed, except for channel connections. Multiple LAN adapters may be needed to avoid LAN bottlenecks. A more powerful CPU and more RAM may be needed if Host Integration Server computers must support significant additional loads, such as file sharing.
Up to 200 users for each server (heavy use) or up to 500 users for each server (light or intermittent use)	A Pentium computer, with at least 128 MB of RAM. A more powerful CPU and more RAM may be needed if Host Integration Server computers must support significant additional loads, such as file sharing.
Up to 25 users for each server	A Pentium computer, with at least 128 MB of RAM. A more powerful CPU and more RAM may be needed if Host Integration Server computers must support significant additional loads, such as file sharing.

See Also

Concepts

Variables Affecting Hardware

Best Practices

Best Practices

Several rules of thumb can be applied when selecting your Host Integration Server hardware:

- Try to avoid memory paging.

  It is useful to increase the amount of physical memory in a Host Integration Server computer to avoid paging. This guideline is true for any Windows Server that performs communications-related functions.

- Increase memory when using Host Print service.

  When using Print service, you may need more memory on the Host Integration Server computer if large files or many print jobs are submitted. You also need adequate disk space to temporarily store the spooled print jobs.

- Use TCP/IP instead of Microsoft Networking.

  The Windows Sockets (Winsock) and TCP/IP combination uses less memory on the Host Integration Server computer than Microsoft Networking or other named socket solutions. TCP/IP also provides better performance than Microsoft Networking because of its lower network overhead.

See Also
Concepts
Variables Affecting Hardware
Estimating Hardware Demands

Planning 3270 Connectivity

In the hierarchical SNA network model most frequently associated with a mainframe computer, you access centralized applications from remote terminals across a network.

This network model uses the information display protocol for IBM mainframe computers known as 3270. This protocol facilitates conversations between the mainframe and devices such as terminals, printers, and controllers. Through the definition and assignment of 3270 logical units (LUs), Host Integration Server provides access to these mainframe resources.

Once you establish the physical connection from the Host Integration Server computer to the mainframe, you need to determine the type of 3270 connectivity your users need. The topics in this section detail the networking services Host Integration Server can provide to your users and offers information on ways to set up these services.

In This Section

3270 Access

Host Integration Server provides 3270 connectivity through 3270 logical units (LUs). A 3270 LU is known as a dependent LU because it requires a mainframe to function. Each 3270 LU defined within Host Integration Server is configured to use an existing connection to the mainframe system. Each 3270 LU corresponds to a matching LU resource allocated on the host computer, usually specified within VTAM. The 3270 LU definition in Host Integration Server is identified by a number that matches the number of the corresponding LU resource on the mainframe, and by a user-specified name.

The 3270 LU is further classified by the type of service provided over the connection. Like physical units (PUs) physical units, LU types are designated by numbers. For example, 3270 display data streams are known as LU 2 streams. Within Host Integration Server, a 3270 LU can be configured as one of the following types:

- Display (LU 2)
- Printer (LU 1 or LU 3)
- Application (LUA)
- Downstream

Once configured, these LUs are accessed from an end-user applications using Host Integration Server client software that is installed on the client workstation. The client software manages communications between a 3270 application (like a terminal emulator) and the Host Integration Server computer. Applications designed for the Host Integration Server client API use the LUs defined within Host Integration Server to establish a communications link from the client personal computer to the mainframe via Host Integration Server.

The link between the LU definition in Host Integration Server and the host LU resource is called a *session*. Sessions can be permanent and automatically started during initialization, or established on an as-needed basis. Concurrent sessions can share the same physical devices and communications links.

See Also
Concepts
Using LU Pools
Assigning LUs to Workstations
Providing Hot Backup and Load Balancing

Deployment Strategies

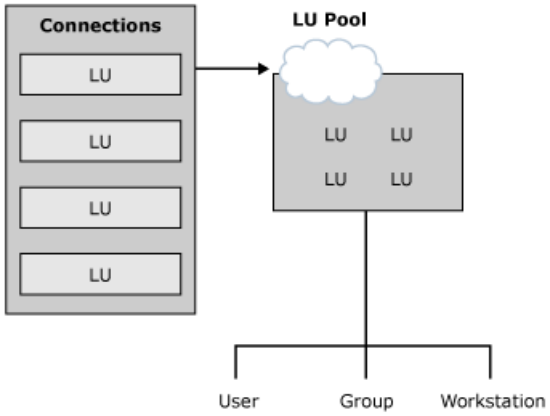
This section describes deployment strategies that can be applied if you are providing 3270 connectivity services including:

- Using LU Pools
- Assigning LUs to Workstations
- Providing Hot Backup and Load Balancing

Using LU Pools

Although you can create individual LUs and assign them to users and groups, using LU pools to manage and deploy a large number of LUs is a more efficient method of administering these resources. LU pools are groupings of LUs that allow you to maximize access to these LUs. As shown in the following illustration, a user, an application, or a downstream system can access the LUs as long as any LU assigned to the pool is free. If any one of the pooled LUs ceases to function, another free LU in the pool is automatically used.

Diagram of creating and assigning LU pools



Creating and assigning LU pools

LU pools also allow groups of intermittent users to use a limited number of host resources more efficiently. Dedicating LUs to specific users who occasionally require host access wastes host resources. Using a pool, you can assign a smaller number of LUs to a group of users who require sporadic access. For example, if a group of 100 users require host access 25 percent of the time, assigning a pool of 25 LUs to the group may fulfill their needs.

See Also

Concepts

Assigning LUs to Workstations

Providing Hot Backup and Load Balancing

Assigning LUs to Workstations

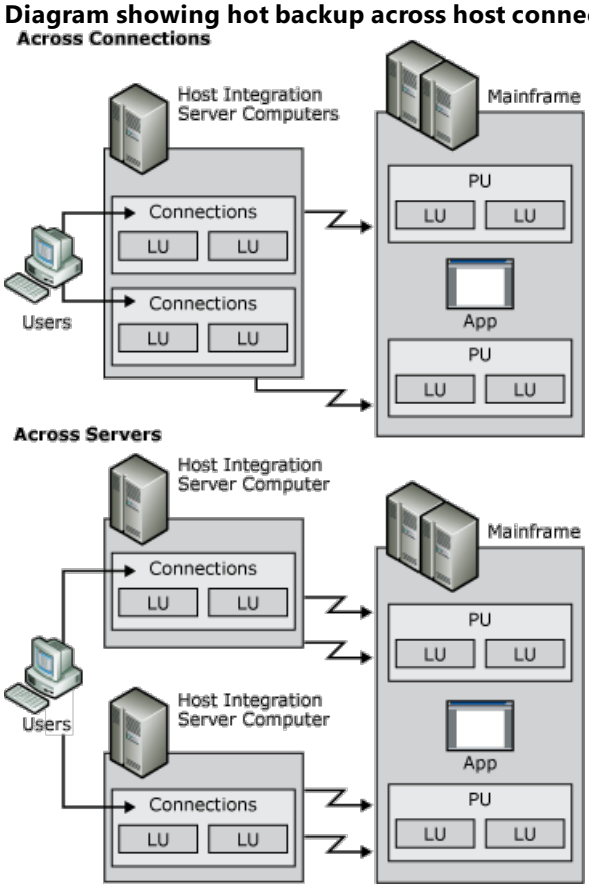
LUs and LU pools can also be assigned to workstations rather than users, effectively locking LUs to a specific machine. Assigning LUs to a workstation makes it easier for users to find and access different resources. For example, 200 hospital employees can share 50 workstations, each of which has a printer attached. Employees may want to access any of the 50 workstations and 50 printers. Instead of assigning 50 printer LUs to a pool and making the pool available to each user, each of the workstations can be added to Host Integration Server and a printer LU can be assigned to each workstation. Now, when a user logs onto any of the 50 workstations, the printer that is attached to the workstation will be available in the list of LUs.

See Also
Concepts
Using LU Pools
Providing Hot Backup and Load Balancing

Providing Hot Backup and Load Balancing

The following diagram shows hot backup across host connections on the same server and across servers.

Diagram showing hot backup across host connections on the same server and across servers



To recover from situations where a particular host connection has failed, the Host Integration Server hot backup feature can be configured to allow for similarly configured resources to automatically fill in and support functions dependent on the failed connection. When a failure occurs, a user can simply reconnect to a given resource using an alternate connection or server without reconfiguring client software. Hot backup can be implemented across host connections on the same server, or across several servers in a domain using LU pools as shown in the preceding figure. Implementing fault-tolerant connections is a recommended strategy for enterprises of any size, and helps to provide reliable host access to your users.

Related to hot backup is a feature called load balancing. Load balancing evenly distributes sessions across multiple host connections and multiple servers using 3270 LU pooling. Instead of explicitly requesting specific LUs, users request the first available LU in a pool and Host Integration Server randomly assigns the user a free LU in the pool. Because each LU can be configured using different connections or servers, the server load can be spread out across all configured resources. Load balancing is implemented automatically when a pool is configured with LUs from multiple servers or connections.

See Also

Concepts

Using LU Pools

Assigning LUs to Workstations

TN3270 Access

TN3270 is a type of Telnet service that allows access to mainframe computers over a TCP/IP network. Users can connect to mainframes using a TN3270 client and the TN3270 service provided with Host Integration Server.

The TN3270 service supports these protocols:

- TN3270, for display sessions
- TN3287, for printer sessions
- TN3270E, for extended display and print sessions

The TN3270 service uses Host Integration Server features to provide mainframe access and to address issues such as security and redundancy when the data communications path between the client and server contains one or more nonsecured segments.

TN3270 Service

The TN3270 service communicates with Host Integration Server using the LUA (Logical Unit for Applications) API. Because of this, LUA-type connections and LUs must be configured on the server. Once configured, LUA LUs and LU pools can be assigned to the TN3270 service and made available for use by TN3270 clients requesting mainframe access.

See Also

Concepts

Setting Port Numbers
Deploying Hot Backup and Load Balancing
Assigning LUs to an IP Address

Deployment Strategies

This section describes deployment strategies that can be applied if you are providing TN3270 connectivity services including:

- Setting Port Numbers
- Deploying Hot Backup and Load Balancing
- Assigning LUs to an IP Address

Setting Port Numbers

As with all TCP/IP services, TN3270 requires a free TCP port in which clients can locate the TN3270 service. By default, the TN3270 service defaults to port 23, the same port as standard Telnet services. Because no two services can share the same TCP port, it is recommended that you change the TN3270 service to use TCP port 24 or some other unused TCP port. When attempting to connect to the TN3270 service from a client application, you must also specify the new TCP port within the application connection settings.

See Also

Concepts

Deploying Hot Backup and Load Balancing

Assigning LUs to an IP Address

Deploying Hot Backup and Load Balancing

A Windows domain can contain one or more Host Integration Server subdomains. Like 3270 LUs, LUA LUs from multiple servers in different subdomains can be assigned to the TN3270 service. This lets you distribute client sessions among the participating servers in the subdomain, thereby balancing the load.

Creating redundant connections to the mainframe and assigning them to a TN3270 service increases service availability. If one server goes down, a client can still access LUA LUs on a different server. Similarly, a single server can be configured with redundant host links to increase hot backup and bandwidth if no other Host Integration Server computers are available.

See Also

Concepts

Setting Port Numbers

Assigning LUs to an IP Address

Assigning LUs to an IP Address

In the same way that you assign 3270 LUs to a user or workstation, you can restrict access to LUA LUs or pools by specifying an IP address or subnet mask for clients that must access the resource. If a workstation has a name that can be resolved using name resolution services like DHCP or WINS, the name can be associated with the resource instead.

Restricting access to clients with specific IP addresses or workstation names increases the security of the LUA resources.

See Also

Concepts

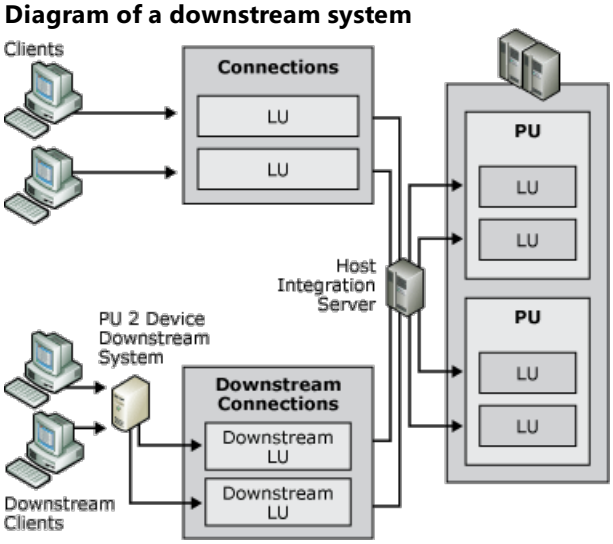
Setting Port Numbers

Deploying Hot Backup and Load Balancing

Downstream Connections

In a hierarchical SNA environment, you configure 3270 communications between SNA nodes using SNA protocols. Usually those nodes are Host Integration Server computers and mainframes. A downstream system, however, is an SNA node that uses Host Integration Server as a physical unit (PU) gateway. To the downstream system, the Host Integration Server computer appears to be the actual mainframe providing the PUs and 3270 LUs. The following figure illustrates a downstream system.

Diagram of a downstream system



A downstream system in this type of environment must be a PU 2 device, for instance, a cluster controller like an IBM 3745, or a client personal computer running a terminal emulator that emulates a PU 2 and acquires LU sessions from the Host Integration Server computer.

On the Host Integration Server computer, two connections are required to support downstream systems:

- A host connection between the server and the mainframe. This can be any standard physical connection method supported by the mainframe.

- A downstream connection between the server and the downstream system. This physical connection can be a DLC 802.2, SDLC, or X.25 connection.

Once configured, Host Integration Server can manage the downstream LUs in a manner similar to other LUs, including assigning them to LU pools.

See Also

Concepts

Deployment Strategies

Deployment Strategies

This type of configuration is useful in environments in which the downstream system may be unable to communicate directly with the mainframe because of hardware or network incompatibilities that the intermediate Host Integration Server computer can resolve.

Using Host Integration Server as a PU concentrator can also help reduce host configuration requirements. LUs from one or more physical units (PUs) can be shared with multiple downstream devices, alleviating the need to configure each downstream device in VTAM on the mainframe system. The result is more efficient use of host resources.

See Also

Concepts

Downstream Connections

Planning APPC Connectivity

A peer-oriented SNA network, using the Advanced Program-to-Program Communications (APPC) protocol, relies on each device in the network to communicate directly with the others. Each computer depends primarily on its own intelligence and does not require constant access to a centrally located host computer.

APPC supports display and other application services across an SNA network. Although peer-oriented SNA networks are usually associated with an AS/400 host system, mainframe systems can also support peer-to-peer networking.

Once you establish the physical connection from Host Integration Server to the AS/400, you need to determine the types of APPC services required by your users. This section discusses the APPC services that Host Integration Server can provide to your users and tells how to set up these services.

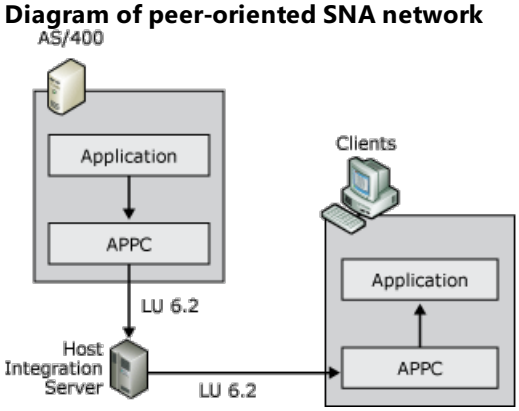
In This Section

Understanding Peer-to-Peer Networking

Devices in peer-oriented SNA networks participate in Advanced-Peer-to-Peer Networking (APPN). Each device, known as a type 2.1 physical unit (PU 2.1), handles all network routing functions, as well as normal computing activities and applications.

LU 6.2 logical units are associated with PU 2.1 devices. Devices in an APPN network appear as LU 6.2 (also called APPC LU) entities. Programs that are executed on these devices are called transaction programs(TPs).

Diagram of peer-oriented SNA network



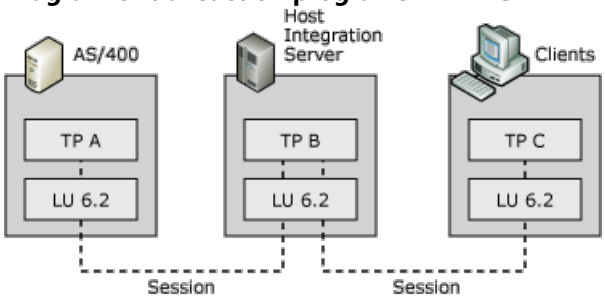
APPC enables TPs on different APPN systems to communicate directly with each other across an APPN network. In APPN networks, Host Integration Server provides support for the APPC protocol and emulates a PU 2.1 low-entry networking (LEN) node.

In the AS/400 environment, APPC is used for a variety of applications, including:

- 5250 access
- TN5250 access
- File transfer

TPs use LU 6.2 names to access other systems and other transaction programs as shown inthe following figure. With Host Integration Server, a transaction program, such as a 5250 terminal emulator, can also use an APPC LU alias to access another TP. In this case, the LU alias maps to an LU name that is actually used to access the other system's TP.

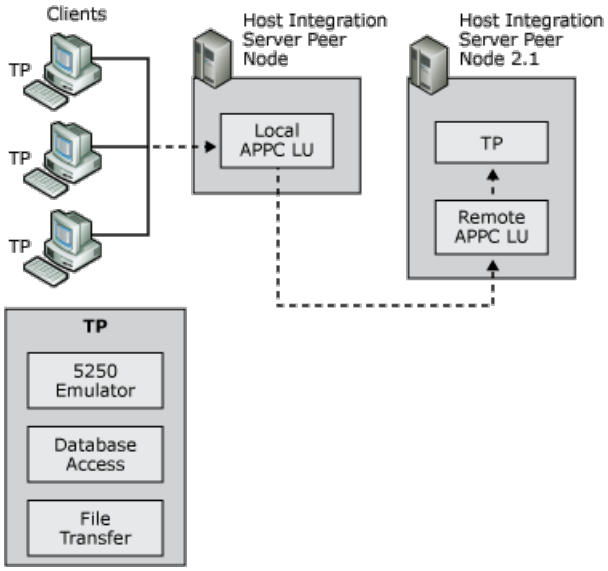
Diagram of transaction programs in APPC



APPC uses pairs of LUs to facilitate simultaneous, bi-directional communication between transaction programs. To achieve this, a local LU and a remote LU are defined on each device in the APPN network.

The perception of local and remote LUs is dependent on the system that you are configuring. When configuring Host Integration Server, the local APPC LU corresponds to the Host Integration Server computer and the remote LU corresponds to the AS/400. Local LUs on one system communicate with remote LUs on another system. If you view the configuration from the AS/400 perspective, the Host Integration Server computer is the remote system and the AS/400 is the local system.

Diagram of conversation components in APPC



When a client/server network TP, such as a 5250 terminal emulator, requests a conversation with a TP on the AS/400 (remote system), the server (local system) acts on behalf of the client request and negotiates an LU 6.2 - LU 6.2 session to the AS/400. The data sent or received from the AS/400 TP is handled by the server and sent to the client TP over the selected client/server protocol. This is illustrated in the preceding figure.

See Also

Concepts

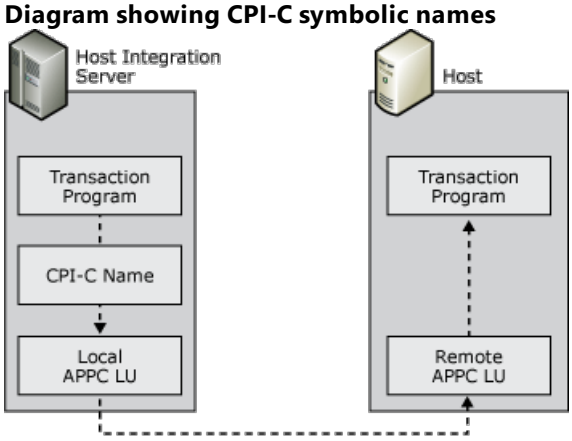
Understanding CPI-C

Understanding CPI-C

Common Programming Interface for Communications (CPI-C) is an application programming interface (API) that uses the LU 6.2 communications architecture. CPI-C comprises a set of C programming language routines that allow applications on computers to communicate with one another to accomplish a processing task, such as copying a file or accessing a remote database.

CPI-C programming provides a mechanism called client-side information that associates a set of parameters with a specified CPI-C symbolic destination name. The CPI-C program uses the symbolic destination name to initialize a conversation using APPC LUs that are associated with the CPI-C symbolic name.

Diagram showing CPI-C symbolic names



Host Integration Server supports the CPI-C API and provides for configuration of CPI-C parameters. These parameters allow applications on the Host Integration Server systems to communicate with applications on any platform that supports APPC communications and CPI-C including mainframes, AS/400s, Windows systems, and UNIX systems.

See Also

Concepts

Understanding Peer-to-Peer Networking

APPC Applications

Advanced Program-to-Program Communications (APPC) can be used to support a wide range of applications. This section describes how Host Integration Server is used to support the following services:

- 5250 terminal access
- TN5250 terminal access
- APPC file transfers

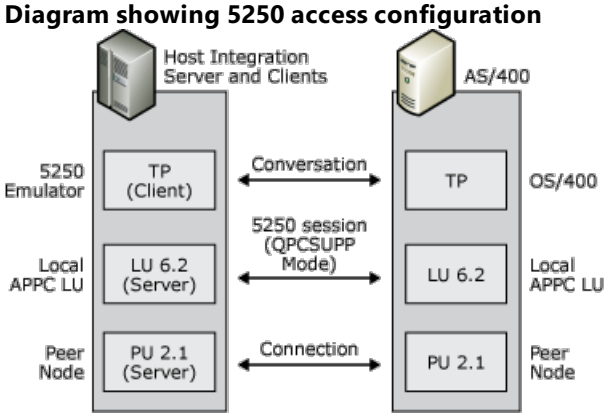
See Also
Concepts
5250 Access
TN5250 Access

5250 Access

If your enterprise contains AS/400 systems, display sessions are provided through APPC using the 5250 data stream. Host Integration Server computers provide APPC access to an AS/400 using 5250 emulation clients. Clients can only communicate with AS/400s using APPC.

For 5250 services, the local APPC LU acts as an identifier for local Host Integration Server clients; the remote APPC LU identifies the AS/400 system. The following figure shows the local and remote LUs used for this configuration.

Diagram showing 5250 access configuration



See Also

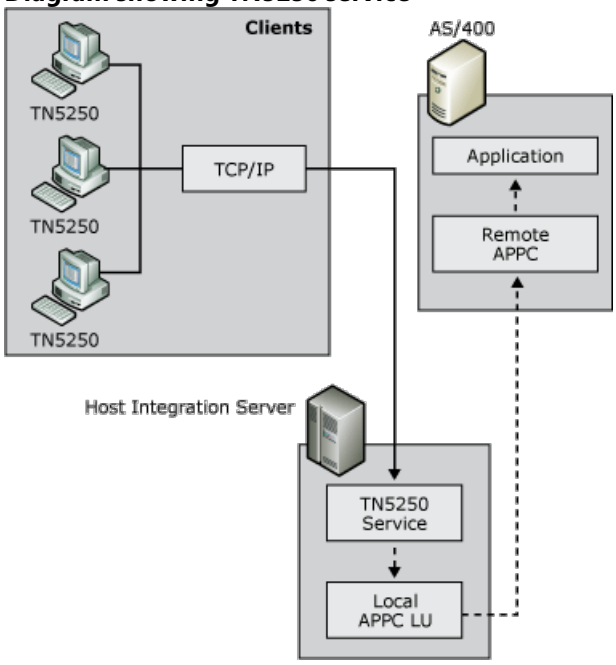
Other Resources

APPC Applications

TN5250 Access

TN5250 is a protocol that allows users to access AS/400 systems over a TCP/IP network using an appropriate TN5250 client terminal emulator. The TN5250 service provided with Host Integration Server enables any TN5250 client to connect to the AS/400 by means of Host Integration Server without installing or configuring TCP/IP on the AS/400. Full 5250 terminal emulation functions are supported by the service, as well as hot backup and security features similar to those provided with the TN3270 service.

Diagram showing TN5250 service



To provide TN5250 access, you need to define local and remote APPC LUs, a mode, an AS/400 user name and password, a terminal type, and, optionally, an IP address and subnet mask.

See Also

Other Resources

APPC Applications

APPC Deployment Strategies

In most cases, deployment strategies are effective for APPC LUs, regardless of the application that the LUs support. This section describes the factors that you should consider when deploying APPC connections with Host Integration Server.

In This Section

Using Independent APPC LUs

Using Dependent APPC LUs

Choosing Modes

Using LU Pools

Configuring LUs

Providing Hot Backup

Choosing IP Settings

See Also

Other Resources

Planning APPC Connectivity

Using Independent APPC LUs

An independent LU can communicate directly with a peer system and does not need the support of a host computer. Independent APPC LUs, as used in AS/400 APPN networks, provide the ability to run multiple, concurrent, parallel sessions between a local and remote LU pair.

When configuring independent APPC LUs, you should note that when Host Integration Server is used to communicate with a transaction program (TP) on a mainframe over an independent APPC LU, the host system must be running VTAM version 3, release 2 (V3R2) or later. The version of Network Control Program (NCP) required on the mainframe depends on the type of front-end processor (FEP) used. For 3725 systems, NCP V4R3 or later is required. For 3745 systems, NCP V5R2 or later is required.

See Also

Other Resources

APPC Deployment Strategies

Using Dependent APPC LUs

A dependent local APPC LU requires the support of a mainframe in order to communicate with a remote TP. Dependent APPC LUs cannot be used to communicate with AS/400s. Unlike independent APPC LUs, dependent APPC LUs only allow a single session per LU.

Dependent APPC LUs are helpful when configuring Host Integration Server to communicate with a mainframe using a version of VTAM earlier than V3R2. Independent LUs are not supported in earlier VTAM versions. Support for dependent APPC LUs is provided with Host Integration Server for compatibility with older VTAM versions. If possible, use independent LUs.

When configuring APPC dependent LUs, you should specify the network name and LU name. Even though they are not required, they are used by software running on the Host Integration Server computer, such as the Windows Event Log. For example, if a remote APPC LU will be partnered with a dependent local APPC LU, naming the remote APPC LU helps to identify any events associated with the LU in the Windows Event Log.

See Also

Other Resources

APPC Deployment Strategies

Choosing Modes

When choosing modes, each LU-LU pair has a mode associated with it that determines session properties for that pair. For independent APPC LU sessions, the Parallel Session Limit parameter is of particular importance because this limit determines the number of simultaneous conversations that a session can support. If you plan to use a remote APPC LU that supports parallel sessions, it can only be used with a mode whose parallel session limit has a value greater than 1.

The following table lists the modes supplied with Host Integration Server and describes the scenarios in which each mode can be used.

Mode Name	Suitability
#BATCH	Batch-oriented sessions
#BATCHSC	Batch-oriented sessions that employ a minimal level of routing security
BLANK	Sessions using a default node name, encoded as eight blank EBCDIC spaces in a BIND APPC command
#INTER	Interactive sessions
#INTERSC	Interactive sessions that employ a minimal level of routing security
QPCSUPP	All sessions with an AS/400 computer
QSERVER	Database connectivity with an ODBC driver

See Also

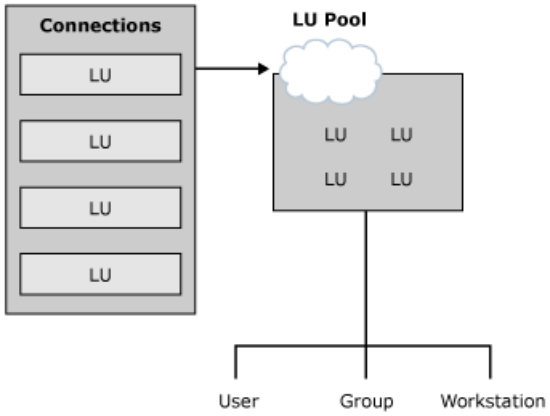
Other Resources

APPC Deployment Strategies

Using LU Pools

Although you can create individual LUs and assign them to users and groups, using LU pools to manage and deploy a large number of LUs lets you administer these resources more efficiently. An LU pool is a grouping of LUs that allows you to maximize access to these LUs as shown in the following figure. A user, an application, or a downstream system can access the LUs as long as any LU assigned to the pool is free. If any one of the pooled LUs ceases to function, another available LU in the LU pool is automatically used.

Diagram showing LU pools



LU pools also allow groups of intermittent users to use a limited number of host resources more efficiently. Dedicating LUs to specific users who occasionally require host access wastes host resources. Using a pool, you can assign a smaller number of LUs to a group of users who require sporadic access. For example, if a group of 100 users require host access 25 percent of the time, assigning a pool of 25 LUs to the group may fulfill their needs.

See Also

Other Resources

APPC Deployment Strategies

Configuring LUs

When a user configures a 5250 emulator to access an AS/400, the emulator is configured with the local and remote APPC LU alias names. The LU alias names are mapped to LU names that are used for the conversation with the AS/400.

Host Integration Server allows you to define a default local and remote APPC LU for each user group accessing the AS/400. Setting default values relieves the user from having to remember APPC LU names; these values should be specified, if possible.

When creating local APPC LUs, it is recommended that you use the name of a user or group as the local LU alias. When the session from a particular user or group is active, the LU alias name is displayed in SNA Manager. Matching local LU alias names with user names allows you to tell which 5250 users are connected to the AS/400. This also makes it easy to keep track of which local LU a particular user should use.

To use this method, create a separate local APPC LU with an LU alias that matches the user name of each person or group being added to the SNA subdomain. After the LU is created, assign the LU to the user as the default local APPC LU along with a default remote APPC LU.

In Host Integration Server, you can also specify an implicit incoming remote LUthat defines the properties to use when Host Integration Server receives a request to start a session with a local LU, and the remote LU named in the request is not recognized by Host Integration Server. You can also specify an implicit incoming mode that defines the session characteristics when a mode is not recognized by Host Integration Server.

If you want to accept an incoming request that can arrive by many different remote LUs without explicitly defining each remote LU, you can specify implicit incoming remote LU pairs, along with their mode. When the implicit incoming remote LU and mode are specified, the remote LU does not need to be recognized by the server. As long as the local LU specified in the session request is recognized, the connection can be initiated.

See Also

Other Resources

APPC Deployment Strategies

Providing Hot Backup

In the AS/400 environment, Host Integration Server uses a combination of LU names and LU aliases over one or more servers to achieve transparent connections with hot backup to an AS/400.

Hot backup connections to an AS/400

A single Host Integration Server computer can use multiple connections to provide hot backup. To use this method, two or more APPC connections to the same AS/400 are configured and appropriate LUs are grouped into a pool. If one of the connections fails, the clients can reconnect to the AS/400 without reconfiguration.

Host Integration Server can also use multiple Host Integration Server computers as backups to one another. If one of the servers fails, the clients are shifted from the failed server to a working server. Use a combination of connections and servers with hot backup to maintain a high level of host availability in your enterprise.

See Also

Other Resources

APPC Deployment Strategies

Choosing IP Settings

IP settings assigned to the TN5250 definitions allow TN5250 clients to connect to the AS/400. By default, the TN5250 definition is not assigned an IP address or a subnet mask. This allows any TN5250 client to connect to the AS/400.

You can restrict access to the TN5250 service by specifying the IP address or subnet mask of the client workstation(s). When these values are specified, only clients whose IP and/or subnet mask match those specified in the TN5250 configuration are allowed access to the AS/400 through the TN5250 service. You can also specify the workstation name in place of the IP, and use a WINS, DHCP, or other name-resolution service to resolve a friendly name to an IP address.

See Also

Concepts

Host Integration Server (SNA) Remote Access Service

Deployment Strategies

Host Integration Server (SNA) Remote Access Service

SNA Remote Access Service integrates the LU 6.2 transport of Host Integration Server with Windows Remote Access Service, allowing administrators to create virtual LAN connections between Windows systems across an existing SNA network. Using SNA Remote Access Service, the SNA network acts as a network backbone, passing network traffic between the Windows systems that are bridged with the host system.

The functions available with SNA Remote Access Service are the same as those for Remote Access Service over ISDN or X.25, except for the dial-back connection feature, which is not supported by SNA Remote Access Service. The SNA Remote Access Service supports the either the Windows Remote Access Service Server or Windows Remote Access Service client, depending on whether the machine to which SNA Remote Access Service is installed is a Host Integration Server client or server. In addition, a Windows Workstation computer running Host Integration Server client software and SNA Remote Access Service can dial out through the Host Integration Server computer that is also running SNA Remote Access Service.

SNA Remote Access Services is configured using local and remote APPC LUs that are assigned to clients using SNA Remote Access Service to connect to mainframe and AS/400 computers. After SNA Remote Access Service is installed, a SNA Remote Access Service port must be specified for each client or server connection that the computer will support. For example, to support one client and four server connections, a total of five ports must be specified. Each port must be added and configured before it can be used by SNA Remote Access Service.

See Also
Concepts
Choosing IP Settings

Deployment Strategies

Since legacy SNA networks may include smaller-bandwidth links like SDLC, and LAN traffic typically generates more network traffic that can be effectively handled by slower connections, care should be taken to resolve bandwidth needs before deploying SNA Remote Access Service in your enterprise.

Before Windows Remote Access Service clients can connect through SNA Remote Access Service, Host Integration Server must be configured to provide the following items for each client system:

- A peer incoming connection
- A remote APPC LU for each client

Each Host Integration Server computer must also be configured with at least one local APPC LU that can be used by SNA Remote Access Service when communicating with the client system. One local LU is required for each SNA Remote Access Service port that is used. For each connection that you set up for use by SNA Remote Access Service, a corresponding remote APPC LU must be configured. This remote LU defines the local LU on the Remote Access Service client that will be used to connect to the server via SNA Remote Access Service. Once the LUs are configured, each remote user must be given permission to dial in to the server.

If you create a SNA Remote Access Service connection between a client and a server that are also connected by a LAN, you must disable the bindings between the Remote Access Server Service and the network adapter to prevent Windows Remote Access Service from attempting to insert the Remote Access Service client into the LAN. This scenario usually occurs when testing a Remote Access Service connection over a network.

After the APPC LUs are configured, the final step is to create a phone book entry for each connection. If you have configured more than one port for use by SNA Remote Access Service, you can create an additional set of phone book entries by associating the same connection with the additional Remote Access Service ports.

A common use of SNA Remote Access Service is to enable an administrator to manage Host Integration Server computers remotely via the SNA network itself. An administrator working at a Windows Workstation running SNA client software and SNA Remote Access Service can use the service to connect to and administer servers at branch locations connected to SNA network.

See Also
Concepts
Choosing IP Settings

Planning for Transaction Integrator

Before installing and using Transaction Integrator (TI), determine whether your mainframe-based transaction programs (TPs) can be used with TI and whether any of them need modification. Answer the following three questions to find out whether TI can invoke your TP:

- Is the TP irretrievably terminal-oriented, or can you expose a request-response interface?
- What programming model does the TP need?
- Does the TP use data types that TI supports?

To use TI to invoke a mainframe-based transaction program (TP), you must separate the business logic from the presentation logic in the TP. TI uses the request-response model; it does not support conversational or pseudo-conversational transactions. The TP must support the so-called ping-pong request-reply mode.

Although TI does not support screen scraping, it does support eight other communication models. Some CICS and most IMS transactions that expose terminal interface can also be invoked using one of the eight supported models. For example, a CICS transaction might be terminal-oriented but still have the business logic partitioned in a separate link model transaction for load balancing or maintainability.

See Also
Concepts
Communication Models
Choosing the Appropriate Programming Model
Other Resources
Data Types
Host and Automation Data

Communication Models

Transaction Integrator (TI) supports the following communication protocols for interacting with the host computer. These protocols are limited to singlerequest/single response. However, the request and response can consist of multiple data segments in the case of unbounded recordsets.

- **CICS TCP Transaction Request Message Link**

  The host transaction must be IBM DPL-enabled. In other words, the mainframe transaction must be designed to be invoked by EXEC CICS LINK.

- **CICS TCP Transaction Request Message User Data**

  The host transaction must use explicit TCP SEND/RECEIVE.

- **CICS LU 6.2 Link**

  The host transaction must be IBM DPL-enabled (Distributed Program Link), that is, the mainframe transaction must be designed to be invoked by EXEC CICS LINK.

- **CICS LU 6.2 User Data**

  The host transaction must use explicit APPC SEND/RECEIVE.

- **IMS Connect**

  Enables you to use TCP/IP with your IMS-based TP without recompiling it. By using IMS Connect or OTMA, you can connect to existing IMS transactions without linking listeners to the TPs.

- **IMS Implicit**

  The IMS program must use implicit message queue and IMS library CBLADLI (rather than CBLTDLI).

- **IMS Explicit**

  The host transaction must use explicit TCP SEND/RECEIVE.

- **IMS LU 6.2 User Data**

  The IMS program must use an implicit message queue (the common design model).

- **OS/400 Distributed Program Calls**

See Also
Concepts
Planning for Transaction Integrator

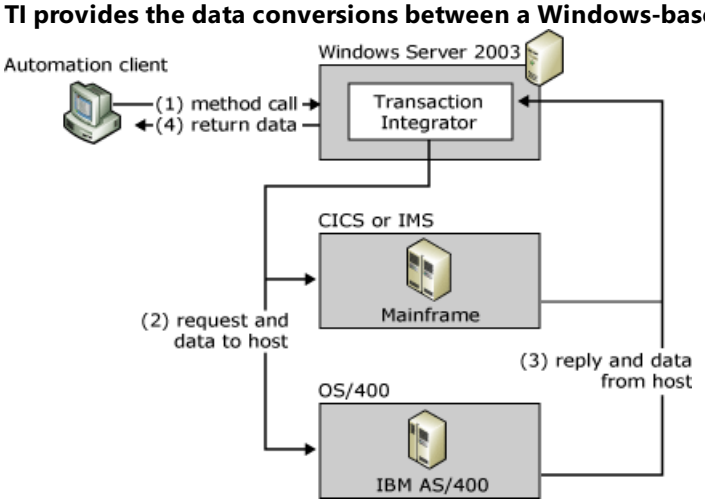
Transaction Integrator Architecture

Transaction Integrator (TI) integrates the IBM Customer Information Control System (CICS), Information Management System (IMS) transaction programs (TP), and OS/400 applications with the Component Object Model (COM) and the .NET Framework by doing the following:

- Creating interfaces to the CICS, IMS, and OS/400 transaction programs.
- Invoking those transactions on the mainframe or midrange computer (also called the host computer) from a Microsoft Windows–based application.

The primary function of TI is to manage the process and data conversions necessary to allow input data to be provided to the host TPs from a COM or .NET Framework application and to send any output data generated from the TP to the Windows-based application. TI provides data type conversion, tabular data definition, and code page translation. The following figure shows an overview of the role that TI plays in the communications between the application and the host.

TI provides the data conversions between a Windows-based application and a host



An example of this type of distributed application is reading a DB2 database on the mainframe to update data in a SQL Server database on Microsoft Windows 2000 Server.

Because all TI processing is done on the Windows 2000 Server or Windows Server 2003 platform, no TI-related executable code (or footprint) is required on the mainframe or midrange computer. TI uses SNA (APPC/LU 6.2) and TCP/IP standard communication protocols for all communications between Windows and the host computer. You can use TI Designer to build TI components, and you can use TI Manager to deploy, configure, and manage those TI components and the TI run-time environment.

In This Section

Online Transaction Processing

Most mainframe and midrange mission-critical applications run as online transaction processing (OLTP) applications underthe direction of a transaction-processing monitor, such as customer information control system (CICS) and information management system (IMS). CICS and IMS are widely used in mainframe environments to create distributed OLTP solutions, such as customer-tracking and order-entry solutions. TI integrates CICS and IMS with COM by creating COM and .NET interfaces to the CICS and IMS transactions and by calling for the services of, or invoking, those transactions on the host from a Windows-based application.

A two-phase commit (2PC) protocol is used when a transaction involves multiple programs running on multiple computers. Use the 2PC protocol to verify that eachl computer has completed its part of the transaction.

In This Section

CICS Components

IMS Components

Two-Phase Commit

Windows Transactions vs. Mainframe Transactions

See Also

Other Resources

Transaction Integrator Architecture

CICS Components

Customer Information Control System (CICS) is a mainframe application system that provides components such as a transaction-processing monitor and a transaction-processing manager for a mainframe computer to run online transaction processing (OLTP) applications. You can install CICS on all three mainframe operating systems: Multiple Virtual Storage (MVS), Virtual Storage Extended (VSE), and Virtual Machine (VM). Due to the popularity of MVS, CICS is commonly installed on MVS mainframe computers. CICS extends the capabilities of a batch-only environment by providing the application system components that allow the mainframe computer to run OLTP applications.

CICS can run online applications on the mainframe computer because CICS acts almost like a separate operating system: it manages its own memory address space, runs its own file management functions, and manages the concurrent execution of multiple transaction applications.

To use Transaction Integrator (TI) successfully, you must understand the following CICS components and terminology:

CICS region

Each instance of CICS running on a mainframe computer is defined in Virtual Telecommunications Access Method (VTAM) by using a VTAM application statement. Each CICS instance defined in an application statement is called a CICS region. It is useful to define multiple CICS regions on a single mainframe computer because it allows you to logically group TPs in separate CICS regions and to use at least one CICS region for test purposes.

TP

The transaction program (TP) is the application software that executes under the supervision of CICS and contains the actual programming code necessary to process the business logic. Other terms that refer to a TP are transaction, host transaction program, application program, and program.

Transaction ID

All TPs that run under CICS are invoked by using a unique, four-character transaction identification (TRANID). This may sometimes be confusing because the transaction ID typically is different than the TP name. For example, the TP that handles CICS resource definitions is called Resource Definition Online (RDO), whereas the transaction ID that starts RDO is CEDA.

Program control table (PCT)

The program control table (PCT) is a CICS table that contains a mapping between TRANIDs and their associated TP names. After the TRANID is invoked, CICS starts the TP associated in the PCT with that TRANID.

File control table (FCT)

The file control table (FCT) is a CICS table that monitors which VSAM files are available to TPs. The FCT lists the name and type of VSAM files and valid operations that users can perform on each file. Although CICS can access other types of data stores, such as DB2, it accesses VSAM most frequently.

RDO

The RDO is a CICS TP that allows a CICS systems programmer to define the resources contained in the internal control tables.

Task

A task executes the functions of the TP; every CICS TP performs its functions by using a task. A CICS TP can use a single task or multiple tasks to perform its functions. Each time a TP is invoked, CICS starts the tasks required to perform its functions. CICS is a multitasking environment, which means that more than one task can, and often is, running at the same time.

See Also

Other Resources

Transaction Integrator Architecture

IMS Components

The information management system (IMS) provides a transaction program (TP) Monitor with an integrated TP Manager and hierarchical database. Both the TP Monitor and the database can coordinate transactions with non-IMS TP Monitors and databases.

To use Transaction Integrator (TI) successfully, you must understand the following IMS components and terminology:

IMS region

IMS uses defined regions to perform its functions. The following regions are typically defined in VTAM when using IMS:

- Control region - The main IMS region. It owns all of the databases that IMS transactions access and is responsible for all communications with the databases. It runs continuously and oversees the operation of other dependent regions.
- Message processing region (MPR) - A dependent region used for processing messages. The control region schedules TPs to run in the MPR. You can have multiple MPRs defined on a single mainframe computer.
- Batch message processing (BMP) region - A dependent region used for processing batch operations.

IMS message queue

The IMS message queue is used by TPs to access the MPP region for processing. Each MPP region has an IMS message queue associated with it. Placing application data in the IMS message queue allows the IMS server TP to use standard Get Unique (GU), Get Next (GN), and Insert (ISRT) calls to exchange data with a client application.

Data Language (DL)/I

Data language (DL)/I is the programming language used in traditional IMS environments to access IMS databases. IMS TPs and CICS TPs can be written in many different programming languages, such as COBOL, PL/I, C, VS Pascal, Ada, REXX, or assembler language. However, when any of these TPs needs to access IMS databases, they must use the proper DL/I calls from their application code. Some of the standard DL/I calls are:

- GU. This call retrieves input data to be processed.
- GN. This call retrieves sequential records.
- ISRT. This call inserts data into a database or returns data to an invoking client.

See Also

Other Resources

Transaction Integrator Architecture

Two-Phase Commit

A given business logic operation can involve multiple programs running on multiple computers. In this design, the transaction is not considered complete unless all of the programs involved complete their executions successfully. For these programs to verify that all other programs that are part of the transaction have completed their transactions, they must employ the two-phase commit (2PC) protocol.

The term transaction (or any of its derivatives, such as transactional), might be misleading. In many cases, the term transaction describes a single program executing on a mainframe computer that does not use the 2PC protocol. In other cases, however, it is used to denote an operation that is carried out by multiple programs on multiple computers that are using the 2PC protocol.

The 2PC protocol is so named because it employs the following two phases prior to committing the operation performed:

- Phase 1—Prepare. In this phase, each of the programs involved in the transaction sends a message to the TP Manager, such as Microsoft Distributed Transaction Coordinator (MS DTC), informing the TP Manager that it is ready to and capable of performing its part of the operation. This phase is also known as prepare because the programs are prepared either to commit the changes or rollback the changes. If the TP Manager receives confirmation from each of the programs involved, it proceeds to phase 2.

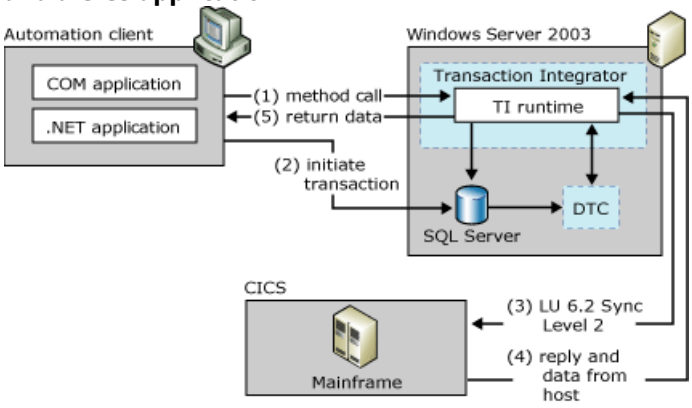
- Phase 2—Commit or Rollback. In this phase, the TP Manager instructs each of the programs to commit or rollback all of the changes that were requested as part of the transaction. A properly executed rollback should return the system to its original state.

Note

The state between phase 1 and phase 2 is known as the in-doubt state. Developers using COM+ or .NET in their applications can decide which parts of the application require access to a TP and which parts do not. TI extends this choice to the mainfra me, as well, by handling calls that require transactions and calls that do not. For applications that require full integration betw een Windows-based two-phase commit and mainframe-based Sync Level 2 transactions, TI provides all the necessary functi onality. TI does this without requiring you to change the client application, without placing executable code on the mainframe , and with little or no change to the mainframe TPs. The client application does not need to distinguish between the TI compo nent (type library) and any other COM+ component reference.

The following figure shows how a Windows-based client application implicitly uses the Microsoft Distributed Transaction Coordinator (DTC) to coordinate the two-phase commit of a distributed transaction involving SQL Server and a CICS TP. DTC coordinates 2PC transactions.

Client application using Transaction Integrator and DTC to coordinate a two-phase commit between SQL Server and a CICS application



Client application using TI and DTC

Two-phase commit (2PC) transactions involve a number of components. To use Transaction Integrator (TI) successfully, you must understand the following 2PC components and terminology:

Sync Point Level 2

TPs can interact with one another by using the LU6.2 protocol at one of three levels of synchronization: Sync Level 0, Sync Level 1, or Sync Level 2. Only one of these three sync levels, Sync Level 2, uses the 2PC protocol. Sync Level 0 has no message integrity, whereas Sync Level 1 supports limited data integrity.

TP Manager

The transaction program (TP) Manager is a system service that is responsible for coordinating the outcome of transactions to be able to achieve atomicity. The TP Managers ensure that the resource managers reach a consistent decision on whether the transaction should commit or abort. The Windows 2000 TP Manager is MS DTC.

Resync service

The LU6.2 Resync Service is a component of Host Integration Server that works with MS DTC to perform automatic recovery to a consistent state as a result of failures at any point in a 2PC transaction. The LU6.2 Resync Service is installed by default when installing Host Integration Server.

Resource Manager

The resource manager is a system service that manages durable data. Server applications use resource managers to maintain the durable state of the application, such as a record of available inventory, pending orders, and accounts receivable. The resource managers work in cooperation with the transaction manager to provide the application with a guarantee of atomicity and isolation (by using the 2PC protocol). Microsoft SQL Server™ and TI are examples of resource managers.

See Also
Concepts
Windows Transactions vs. Mainframe Transactions
Other Resources
Online Transaction Processing

Windows Transactions vs. Mainframe Transactions

In the Host Integration Server Help, a transaction in the Microsoft Windows, COM, COM+, or .NET Framework environments does not mean the same thing as a transaction in the mainframe environment.

- A transaction in the Windows environment is a set of actions coordinated by the Microsoft Distributed Transaction Coordinator (DTC) as an atomic unit of work that meets the ACID test; in other words, a transaction is atomic, consistent, isolated, and durable. Either all the actions in the transaction are completed, or none of them are completed.

- A transaction in the mainframe host (CICS or IMS) environment is a section of code in a structured transaction program (TP), and a TP is a single COBOL program file that contains one or more mainframe transactions. Therefore, a mainframe transaction may or may not meet the ACID test.

A TI Automation server is a TI component deployed in a COM+ or .NET Framework application. A single method in a TI Automation server invokes a single mainframe-based TP. Any TI method in the TI Automation server can invoke any transaction in the TP, but it is the TP that determines which of its transactions to run. The mainframe TP makes this decision based on the information sent to it from the TI Automation server. A CICS or IMS TP can provide any type of service, such as terminal interaction, data transfer, database query, and database updates. A TP can also contain one or more transactions.

A mainframe TP also has a specialized meaning in the IBM CICS environment. Any program that uses Advanced Program-to-Program Communications (APPC) with another program is referred to as a transaction program (TP). APPC is a set of protocols developed by IBM specifically for peer-to-peer networking among mainframes, AS/400s, 3174 cluster controllers, and other intelligent devices. For a TP to communicate directly with another TP using APPC, the two programs must first establish an LU 6.2 session and conversation with each other.

LU 6.2 is the de facto standard protocol for distributed transaction processing in the mainframe environment. It is used by both CICS and IMS subsystems. One program can interact with another program at one of three levels of synchronization:

- Sync Level 0 has no message integrity beyond sequence numbers to detect lost or duplicate messages.

- Sync Level 1 supports the CONFIRM-CONFIRMED verbs that allow end-to-end acknowledgment for client and server.

- Sync Level 2 supports the SYNCPT verb that provides ACID (atomicity, consistency, isolation, durability) properties across distributed transactions by way of two-phase commit (2PC).

Of the three sync levels, only Sync Level 2 provides the same guarantees provided by a Windows, COM, COM+, or .NET Framework transaction.

Note
The TCP/IP protocol is not designed for distributed transaction processing, so TCP/IP does not provide the ACID guarantee th at 2PC in LU 6.2 Sync Level 2 provides. Therefore, it is the network protocol (LU 6.2 or TCP/IP) that determines whether it is p ossible to guarantee that a transaction in a TP operates as an atomic, consistent, isolated, and durable unit.

Thus, in the CICS and IMS environment, the term transaction program (TP) may or may not imply the use of 2PC. The term transaction program refers to the program itself. It is only when the term transaction is qualified by adding the term Sync Level 2 that the Windows developer and the mainframe developer can be sure they are referring to the same thing.

TI supports both Sync Level 0 and Sync Level 2 conversations over LU 6.2 in SNA networks. If a method invocation is part of a DTC-coordinated transaction, TI uses Sync Level 2 to communicate with CICS or IMS version 6.0 with Resource Recovery Services (RRS). If a method invocation is not part of a DTC-coordinated transaction, then TI uses Sync Level 0.

See Also
Concepts
Support for Transactions and Two-Phase Commit

Transaction Integrator Basic Functions

In current Internet-driven, graphical user interface (GUI) computing environments, users usually prefer to access mainframe online transaction processing (OLTP) applications using the same interfaces that they use to access the Internet or their organizations intranet, instead of by using traditional green screen, dumb terminal access methods. Transaction Integrator (TI) provides any application that is compatible with Microsoft Windows 2000 Server or Microsoft Windows Server 2003 access to transactions and data in CICS, IMS, OS/400 applications. By providing this access, TI reduces the time and effort involved in programming specialized interfaces for mainframe computers.

As a generic proxy for a mainframe or midrange computer, TI intercepts object method calls and redirects those calls to the appropriate host application. TI also handles the return of all output parameters and values from the host computer. When TI intercepts the method call, it converts and formats the parameters from the representation understood by the Microsoft Windows platform into the representation that is understandable by the host transaction program (TP).

In This Section

Managing Data Input/Output

Data Type Conversion

Tabular Data Definition

Code Page Translation

See Also

Other Resources

Transaction Integrator Architecture

Managing Data Input/Output

Most mainframe and midrange transaction programs (TP) rely on a set of input parameters that carry input data, typically provided by a user at a dumb terminal, to the TP. TPs then perform some function with the input data and return output data through their defined output parameters. The primary function of Transaction Integrator (TI) is to manage the process and data conversions necessary to allow input data to be provided to the mainframe TP from a COM or .NET Framework application and to send any output data generated from the TP to a COM or .NET Framework application. To accomplish this, TI provides data type conversion, tabular data definition, and code page translation.

See Also

Other Resources

Transaction Integrator Basic Functions

Data Type Conversion

One of the primary features of Transaction Integrator (TI) is converting and formatting a method's data from the data types understood by the Windows platform into the data types understood by a mainframe transaction program (TP). The conversion is defined at design time and implemented at runtime. At design time, the developer uses the TI Designer to associate a COM or .NET data type with a COBOL or RPG data type. TI provides default mappings between standard COM or .NET data types and COBOL or RPG data types, and the developer can either accept the default mappings or override the default with other mappings supported by TI. The TI Designer records the mappings in the TI component library, and the generated COBOL or RPG data declarations reflect them.

See Also

Reference

Converting Data Types from Automation to OS/390 COBOL
Converting Data Types from OS/390 COBOL to Automation
Converting Data Types from RPG to Automation
Converting Data Types from Automation to RPG

Other Resources

Transaction Integrator Basic Functions

Tabular Data Definition

In many cases, the input or output data that Transaction Integrator (TI) handles might be in tabular or array format. TI allows you to define this type of data as one of the following formats:

- Recordset. A recordset provides a means of presenting and manipulating tabular data in a Microsoft ActiveX® Data Objects (ADO) environment. A recordset contains all of the ADO information to make it manageable by any ADO application. A recordset is the primary object used for retrieving and modifying tabular data by using ADO. A recordset object represents a set of records in a table. Recordsets allow TI to support what is effectively an array of a structure (or table in COBOL terminology); it even can support the special case of a structure that is a recordset containing only one row. Each column in the row can contain only a single data element. Recordsets cannot be nested or contain arrays.

- User-defined type (UDT). Unlike recordsets, which must contain all of the formatting necessary to expose them to ADO applications, a UDT is just raw data and can therefore be faster than recordsets. A UDT can contain an ordinary (fixed-size) array. It can also contain a dynamic array. You can combine variables of several different types to create UDTs. UDTs are useful when you want to create a single variable that records several related pieces of information.

- Array. In the COM/COM+ and .NET environments, arrays are SAFEARRAYs that contain information about their bounds and contain the data for the array elements. SAFEARRAYs are mapped to fixed-size arrays on the host computer. SAFEARRAYs have a variable size and require custom information to be marshaled to and from fixed-size arrays on the host computer.

Arrays are created on the mainframe computer during the import process when a simple data type has one or more OCCURS clauses. The OCCURS clause can represent a fixed or variable-length table. Although it is possible in COBOL to have nested OCCURS DEPENDING clauses, only the OCCURS DEPENDING length specifier for the outermost table dimension is supported by TI. The TI Designer ignores nested length specifiers.

Note
A UDT and recordset that have the same fields look the same in COBOL.

See Also
Other Resources
Transaction Integrator Basic Functions

Code Page Translation

The choice of language or code page in TI Manager for a remote environment (RE) determines which code page is used to convert from Unicode on the automation side to Extended Binary Coded Decimal Interchange Code (EBCDIC) on the mainframe side. You can either select the language and accept the subsequent default code page for that language, or you can select the specific code page itself.

See Also

Other Resources

Transaction Integrator Basic Functions

Transaction Integrator Components

Transaction Integrator (TI) consists of three major components: two graphical user interfaces, TI Designer and TI Manager, and the TI run-time environment that automatically handles the actual transaction integration.

In This Section

TI Designer

TI Manager

TI Runtime

TI Designer

TI Designer is a design-time development tool that is hosted inside, and uses the graphical user interface, of Microsoft Visual Studio. You use TI Designer to set the rules for mapping and resolving data types between a Windows resource (such as COM+, the .NET Framework, BizTalk Server) and an IBM batch or online transaction program.

TI Designer creates TI components stored as type libraries (.tlb files) or .NET Framework assemblies (.dll) describing the methods and data for a mainframe TP. Each TI component includes TI run-time environment settings that associates the component with a specific type of remote environment (RE) class, for example CICS LU 6.2 Link, for the mainframe or midrange host computer. In addition to the description of the methods and Automation parameters, the TI component type library contains custom data describing how data types are mapped between a method in TI Automation server and a mainframe transaction in a COBOL transaction program (TP) or a OS/400 transaction in a RPG transaction program. It also establishes other parameters that affect data conversion at runtime.

TI components are not true COM or .NET Framework components because they are type libraries. However, when you place a TI component in a COM+ or .NET Framework application, the TI component becomes encapsulated in a true COM component or .NET Framework assembly, and the COM+ or .NET Framework application becomes a TI Automation server that can be used by any COM-based or .NET Framework application.

Note
When you create a class in Visual Basic, the component type library is embedded in the .dll file that also contains the logic for the methods. In other words, unlike TI, the type library is not in a separate .tlb file. In the case of TI, the .dll file is a generic .dll file that holds the TI run-time environment. Together, the .tlb file created by TI Designer and the generic .dll file installed by TI are equivalent to the .dll file created for a Visual Basic class.

You can also use TI Designer to import or export the data definition section of the COBOL or RPG code making up mainframe transaction programs (TPs). TI Designer can automatically generate a TI component (type library) from imported COBOL code, or it can export generated COBOL code that you can use in your mainframe TP.

See Also

Other Resources

Transaction Integrator Components

TI Manager

TI Manager a design-time administrative tool that is hosted inside, and uses the graphical user interface, of the Microsoft Management Console. You use TI Manager to configure end-points, associate resources with requests, and define security and access rules. The rules and mapping of the data transformations are created in the Transaction Integrator (TI) Designer. TI Manager configures, administers, and manages TI components and the TI run-time environment settings they contain.

TI Manager gives you direct access to Windows Component Services (COM+). By using the Component Services folder in TI Manager, you can configure and administer COM components and COM+ applications, configure your system, deploy components, and configure and monitor services. For example, you can use COM+ to:

- Create new COM+ applications.
- Deploy TI components in COM+ applications.
- Administer application security.
- Administer distributed transactions by using Microsoft Distributed Transaction Coordinator (MS DTC).
- View transaction statistics.
- Resolve transaction states.
- Configure routine component and application behavior, such as participation in transactions.

For more information about application development with Component Services, see COM+ (Component Services) under "Component Services" in the Microsoft Platform SDK.

See Also

Other Resources

Transaction Integrator Components

Transaction Integrator Manager Help

TI Runtime

The TI run-time environment is a specialized run-time environment started by Windows or a requesting IBM application program when the application contains a TI component. For each TI component you create, the TI run-time environment provides the Automation server interface and communicates with the mainframe programs. The TI run-time environment does not have a visible user interface.

As a generic proxy for the mainframe or AS/400 computer, the TI run-time environment intercepts object method calls and redirects those calls to the appropriate mainframe program. It also handles the return of all output parameters and return values from the mainframe. When TI intercepts the method call, it converts and formats the method's parameters from the representation understandable by the Windows 2000 Server or Windows Server 2003 platform into the representation understandable by host transaction programs (TPs).

The TI Automation Server is a COM object that exposes the functionality of a mainframe TP as an Automation interface method. It can expose all of the TP's functionality, a subset, or a superset. A TI Automation server that supports two-phase commit contains the transaction logic that determines when work will be committed or rolled back. A TI Automation server is a TI component that you have deployed in a COM+ application. A client application calls the TI Automation server to invoke the mainframe TP, pass parameters, and return results.

At run time, the TI run-time environment intercepts method invocations from a client application for a TI component library and provides the actual parameter conversion and formatting.

The client application can be any COM-based or .NET Framework application that calls a TI Automation server to invoke a mainframe TP. The client application provides the presentation layer for the application or data. It can be anything capable of calling a COM+ or .NET Framework object, including an Active Server Page (ASP), a Visual Basic application, or even a Microsoft Office application. The client application that uses a TI Automation server (a TI component embedded in a COM+ application) can be running on computer that is running Windows 2000, Windows XP, any later version of Windows, or any other operating system that supports the distributed Component Object Model (DCOM). DCOM is language-independent, so developers can build their client application by using the languages and tools with which they are most familiar, including Microsoft Visual Basic®, Visual Basic for Applications, Microsoft C#®, Microsoft Visual C++®, Microsoft Visual J++™, Delphi, Powerbuilder, and Microfocus Object COBOL. The client application can then easily make calls to the TI Automation server (or any other Automation object) registered on Windows 2000 Server or Windows Server 2003.

Then the TI run-time environment sends and receives the method calls to and from (in and out of) the appropriate mainframe TP. TI uses the TI component library created in TI Designer at design time to transform the parameter data that is passed between the TI Automation server and the mainframe TP. TI also integrates with Component Services and with Microsoft Distributed Transaction Coordinator (DTC) to provide two-phase commit (2PC) transaction support in SNA networks.

The TI run-time environment uses the information in the TI component (type library) and the associated RE to:

- Activate the TP on the mainframe in the RE.
- Pass the parameters specified by the TI component to the TP on the mainframe by way of the associated RE.
- Run the TP.
- Return the results of the TP to the COM+ application (the TI Automation server, which in turn returns the results to the client application that called it.

This The TI runtime environment provides the proxy that the Automation server uses to invoke the mainframe TP. The TI run-time environment provides these functions:

- Translates between Automation and COBOL data types.
- Translates messages to and from the mainframe.
- Provides a generic object for COM+, the behavior of which is described by a TI component (type library) for a specific instance.

See Also

Other Resources
Programming Models

Programming Models

A programming model defines the method(s) used to access and integrate server applications with host applications. A programming model is a combination of:

- The communication protocol that is used to exchange data with the remote application program.
- The target host environment used to host the server application program.
- The interaction semantics defined by the application to control connect, data exchange, and disconnect sequences.

Transaction Integrator supports a set of predefined programming models for Windows-initiated processing and for host-initiated processing. The following table summarizes the 11 available WIP programming models depending on the protocol and the target environment.

Protocol	Target/Host Environment	Host Integration Server Programming Model	Host Integration Server COMTI name
TCP/IP	CICS	TCP Transaction Request Message (TRM) Link	MS Link
TCP/IP	CICS	TCP Enhanced Listener Message (ELM) Link	n/a
TCP/IP	CICS	TCP Transaction Request Message (TRM) User Data	Concurrent Server
TCP/IP	CICS	TCP Enhanced Listener Message (ELM) User Data	n/a
TCP/IP	IMS	IMS Connect	IMS Open Transaction Management Architecture (OTMA) Connect
TCP/IP	IMS	IMS Implicit	Implicit
TCP/IP	IMS	IMS Explicit	Explicit
TCP/IP	OS/400	OS/400 Distributed Program Calls (DPC)	n/a
LU6.2	CICS	CICS LU6.2 User Data	CICS using LU6.2
LU6.2	CICS	CICS LU6.2 Link	CICS using Link
LU6.2	IMS	IMS LU6.2 User Data	IMS using LU6.2

The following table summarizes the five available HIP programming models depending on the protocol and the target environment.

Protocol	Target/Host Environment	Host Integration Server Programming Model	Host Integration Server COMTI name
TCP/IP	CICS	TCP Transaction Request Message (TRM) Link	n/a
TCP/IP	CICS	TCP Enhanced Listener Message (ELM) Link	n/a
TCP/IP	CICS	TCP User Data	n/a

TCP/IP	OS/400	OS/400 Distributed Program Calls (DPC)	n/a
LU6.2	CICS	CICS LU6.2 User Data	n/a
LU6.2	CICS	CICS LU6.2 Link	n/a

In This Section

TCP Transaction Request Message Link

TCP Enhanced Listener Message Link

TCP Transaction Request Message User Data

TCP Enhanced Listener Message User Data

IMS Connect

IMS Implicit

IMS Explicit

OS/400 Distributed Program Calls

CICS LU6.2 Link

CICS LU6.2 User Data

IMS LU6.2 User Data

Choosing the Appropriate Programming Model

Supported Data Flow Models

Iterative vs. Concurrent TCP/IP Models

See Also

Other Resources

Transaction Integrator Architecture

TCP Transaction Request Message Link

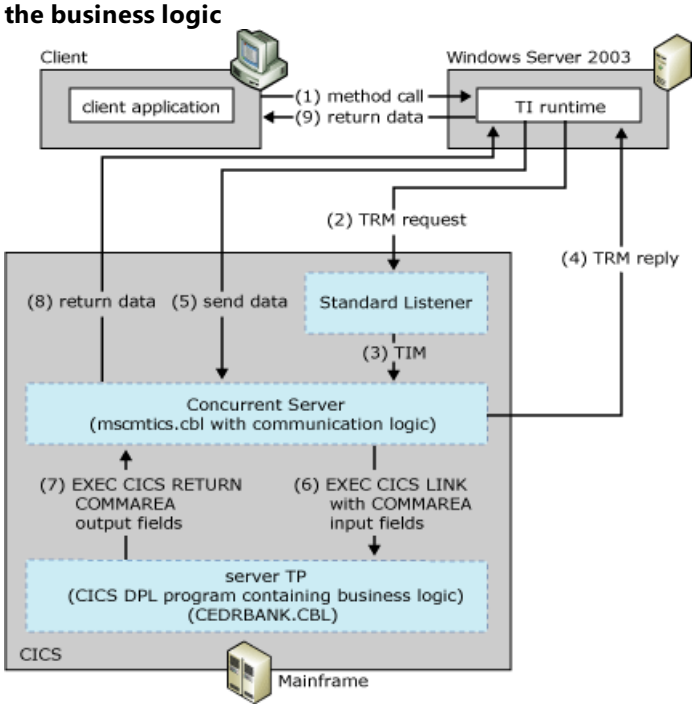
The TCP transaction request message (TRM) Link model allows data and parameters to be passed between TI and the server TP through the COMMAREA. The model also allows a Concurrent Server to Link to a CICS DPL program. The standard Listener for TCP/IP uses two network exchanges to execute a single transaction program and requires the client to:

- Send a Transaction Request Message (TRM) to the standard Listener.
- Receive a TRM reply from the application program.
- Send the application request data stream to the server transaction program.
- Receive the application reply data from the server transaction program.

The TCP TRM Link model is based on the CICS Concurrent Server model. The TCP TRM Link model is a Microsoft variant that supports execution of DPL server application programs within the CICS environment and maintains compatibility with the CICS LU6.2 Link programming model.

The following figure summarizes the workflow occurring between the client, the standard CICS Listener, the Concurrent Server, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Process by which the client starts the default Listener, which passes the call to the concurrent server, which then sends and receives data from the client, which the server then passes to the CICS DPL program for processing by the business logic



Summary Workflow for the TCP TRM Link Programming Model

The TCP TRM Link programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

If the application is a .NET Framework assembly, the TI Automation proxy:

a. Reads in the assembly and meta data created previously by the TI Designer.

b. Maps the .NET Framework data types to COBOL data types.

The TI Automation proxy then:

a. Calls the conversion routines to convert the application data to mainframe COBOL types.

b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. passes the data stream to the TCP transport component.

4. The TI TCP transport sends a connect request to the standard Listener using the Internet Protocol (IP) address of the mainframe computer and the port address of the Listener.

5. The standard Listener accepts the connection request and tells TI runtime to send the TRM. The standard Listener then waits for the TRM.

   The TRM is a formatted data record that identifies the server TP to be invoked by using its TRANID. The Listener TP is a special mainframe TP, whose main function is to receive server TP invocations sent by client applications running TCP/IP.

   The TRANID of the IBM-provided, standard Listener TP is CSKL. The TP name of the Listener TP, as it appears in the CICS program control table (PCT) is EZACIC02.

6. TI runtime formats either a standard or custom TRM and sends it to the standard Listener. TI runtime then waits for the TRM reply.

7. The standard Listener receives the TRM, sends TI runtime a receive confirmation, and then reads the contents of the TRM. The Listener interprets the information in the TRM and extracts the transaction ID of the Concurrent Server program that is to service the request.

8. The standard Listener starts the concurrent server TP program (Mscmtics.cbl sample application) that is identified by the TRANID in the TRM using EXEC CICS Start.

   Mscmtics.cbl is the Microsoft sample TP file that is used to pass data between TI and the server TP using the COMMAREA. The Mscmtics.cbl sample TP is developed by Microsoft and provided as part of the Host Integration Server software. It is located in the $\Microsoft Host Integration Server\SDK\Samples\Comti\ProgrammingSpecifics\Tcp. The code must be compiled, linked, and installed on the mainframe computer prior to using this model.

Note
If the standard Listener is unable to start the Concurrent Server, the Listener formats an error message and sends it bac k to the COMTI TCP Transport. Reasons the Listener might be unable to start include: Rejected connection due to limited CICS resources (for example, exceeds the maximum number of CICS tasks or concur rent server tasks) Invalid or disabled TRANID for the concurrent server Invalid, disabled or unavailable Concurrent Server program associated with the transaction ID

Note

The error message from the CICS listener is character based and always begins with the letters EZY. The length of the error message is variable, and the end of the message is determined by the socket closed by the CICS Listener. The standard Listener calls the socket application protocol interface (API) in the host environment. The standard Listener cannot send the TRM Reply. The TRM Reply represents a synchronization process that allows time for the transaction program to be started prior to the application request data being sent by the client. This synchronization process is necessary due to internal CICS architectural consideration (there is no guarantee as to when a transaction program is started after the request is made).

After the standard CICS Listener has issued the start command for the concurrent server transaction, the standard Listener is no longer needed for application processing and is free to listen for another incoming request.

9. After the concurrent server is running, it reads the transaction initial message (TIM) sent by the standard Listener.

   The TIM describes the TCP/IP environment in which the server is running and contains the TCP/IP socket information the concurrent server uses to communicate with the COMTI TCP Transport and the client message header the concurrent server uses to customize its execution behavior. The header contains the name of the server program to be linked to.

10. The Concurrent Server:

    a. Formats the standard or custom TRM reply.

    b. Sends a TRM reply to the TI TCP Transport to inform it that it can now send the application request data.

    c. Issues a receive and waits for the application request data.

    Sending of the TRM reply completes the 1st part of the standard Listener exchange sequence.

11. TI runtime evaluates the TRM and passes the data to the Concurrent Server program through the CICS COMMAREA by using a standard EXEC CICS Link call. TI runtime also sends a socket (that is, 2 byte) shutdown and then waits for the reply data.

12. After the Concurrent Server receives the application request data it links to the serving application program that was specified in the TRMs client message header. The CICS EXEC CICS LINK command is used to start the real server application. The Link command passes the application data received from the COMTI TCP Transport to the common area of memory (COMMAREA) and performs the business logic on the data. All business logic is defined in the server TP.

13. After the server application program has finished processing the request and formulating the reply, it issues an EXEC CICS RETURN command to give control back to the Concurrent Server (mscmtics.cbl) program. The server TP prepares the reply data along with a standard or custom TRM, accepts the data from the COMMAREA, and then sends the application reply data back to the TI TCP Transport through the COMMAREA. Completing the processing of the application data signals the end of the 2nd exchange sequence.

14. The concurrent server closes the socket.

15. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the TCP transport component.

    b. Reads the message buffer.

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the COBOL data types to the automation data.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

If the application is a .NET Framework assembly, the TI Automation proxy:

a. Maps the COBOL data types to the .NET Framework data types.

b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

16. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

To implement this model, you must provide TI with an IP address, a port number, and a CICS program name to execute the application passed by the concurrent server program (Mscmtics.cbl). The model requires the installation, within CICS, of the IBM-supplied default Listener (EZACIC02). The CICS IBM default Listener uses IBM-provided default settings.

Host Integration Server includes sample code showing how to implement the TCP TRM Link programming model. The sample code is located at *****installation directory***\SDK\Samples\AppInt**. Start Microsoft Visual Studio, open the tutorial of your choice, and follow the instructions in the **Readme**.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: CICS TCP/IP Socket Interface Guide (IBM Document #SC31-7131).

See Also

Tasks

Transaction Request Messages

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components

Choosing the Appropriate Programming Model

Other Resources

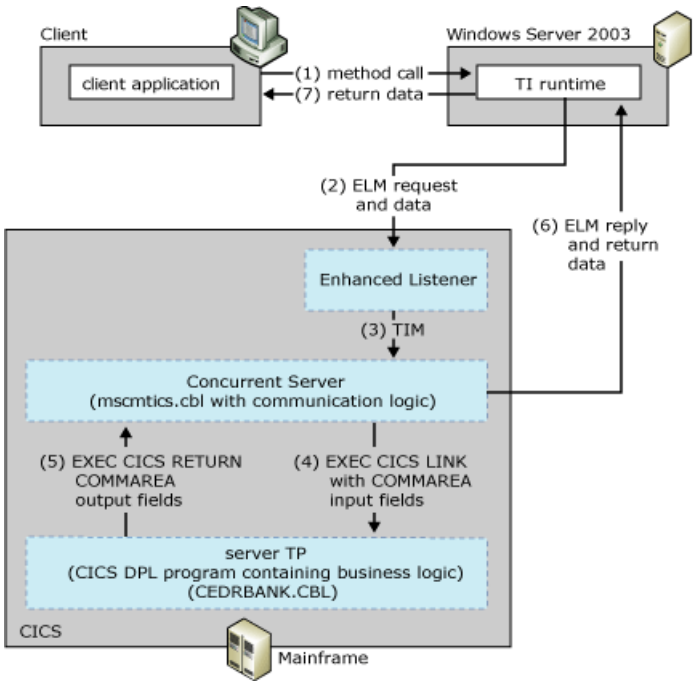
Transaction Integrator Components

TCP Enhanced Listener Message Link

The TCP enhanced listener message (ELM) Link model allows data and parameters to be passed between TI and the server TP using the COMMAREA. The model also allows a Concurrent Server to link to a CICS DPL program. The enhanced Listener was introduced in CICS Transaction Server version 1.4, and its architecture increases the efficiency of the CICS TCP/IP environment by eliminating the TRM and TRM Reply sequence sequence. The enhanced Listener accepts a header and request data from the client in the initial stream and eliminates the need for the server application to deliver a separate response before the application data is made available. The enhanced Listener requires the client to:

- Construct and send a single data stream composed of a request header followed by the application request data to the server application program
- Receive a single data stream that consists of a reply header and application data from the server application program

The following figure summarizes the workflow occurring between the client, the enhanced CICS Listener, the Concurrent Server, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.



Summary workflow diagram for the TCP ELM Link programming model

The TCP ELM Link programming model works as follows:

1. An application invokes a method in a TI component configured in Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. reads in the type library created previously by the TI Designer
    b. maps the automation data types to COBOL data types

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. reads in the assembly and meta data created previously by the TI Designer
    b. maps the .NET Framework data types to COBOL data types

The TI Automation proxy then:

a. calls the conversion routines to convert the application data to mainframe COBOL types

b. builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. passes the message to the TCP transport component.

4. The TI TCP transport sends a connect request to the enhanced Listener using the Internet Protocol (IP) address of the mainframe computer and the port address of the Listener.

5. The enhanced Listener accepts the connection request and tells TI runtime to send the ELM. The enhanced Listener then waits for the ELM.

   The ELM is a formatted data record that identifies the server TP to be invoked by using its TRANID. The Listener TP is a special mainframe TP, whose main function is to receive server TP invocations sent by client applications running TCP/IP. The TRANID of the IBM-provided enhanced Listener TP is defined by the user.

6. TI run-time formats the ELM and sends it to the enhanced Listener. TI then bypasses the transport logic that waits for a ELM reply and immediately sends the application request data after the request header. TI then waits for the ELM reply.

7. The enhanced Listener receives the 35 byte ELM, and then reads the contents of the ELM header. The enhanced Listener places the 35 bytes in the transaction initial message (TIM) but does not operate on its content.

   The TIM describes the TCP/IP environment in which the server is running and contains the TCP/IP socket information the concurrent server uses to communicate with the COMTI TCP Transport and the client message header the concurrent server uses to customize its execution behavior. The header contains the name of the server program to be linked to.

8. The enhanced Listener starts the concurrent server TP program (Mscmtics.cbl sample application) that is defined in the Listener Definition.

   Mscmtics.cbl is the Microsoft sample TP file that is used to pass data between TI and the server TP using the COMMAREA. The Mscmtics.cbl sample TP is developed by Microsoft and provided as part of the Host Integration Server software. It is located in the $\Microsoft Host Integration Server\SDK\Samples\Comti\ProgrammingSpecifics\Tcp. It must be compiled, linked, and installed on the mainframe computer prior to using this model.

Note
If the enhanced Listener is unable to start the Concurrent Server, the Listener formats an error message and sends it back to the COMTI TCP Transport. Reasons the Listener might be unable to start include:

- rejected connection due to limited CICS resources (for example, exceeds the maximum number of CICS tasks or concurrent server tasks)
- invalid or disabled TRANID for the concurrent server
- invalid, disabled or unavailable Concurrent Server program associated with the transaction ID

Note
The error message from the CICS listener is character based and always begins with the letters EZY. The length of the error message is variable, and the end of the message is determined by the socket closed by the CICS Listener.

9. The enhanced Listener calls the socket application protocol interface (API) in the host environment. After the enhanced Listener has issued the start command for the concurrent server transaction, the enhanced Listener is out of the application processing loop and is free to listen for another incoming request.

10. The concurrent server retrieves the TIM, connects the socket, and reads the contents of the ELM.

11. TI passes the application data through the CICS COMMAREA to the server application program that contains the business logic using a standard EXEC CICS Link call. TI runtime also issues a shutdown for the sending 1/2 socket and then waits for the reply data.

12. The server TP receives the application data, processes the request, and performs the business logic on the data. All business logic is defined in the server TP.

13. The concurrent server sends the ELM reply header to TI through the COMMAREA.

14. The server TP prepares the reply data and then sends the response to the client through the COMMAREA.

15. The application reply data stream consists of two parts. The first is an ELM reply that informs the transport as to the success or failure of the request. The TCP Transport will consume the ELM reply from the stream and then, if the ELM reply indicates the call was successful, receive the application reply data until the socket is closed by the Concurrent Server.

16. The concurrent server closes the sockets

17. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the TCP transport component.

    b. Reads the message buffer.

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the COBOL data types to the automation data.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

    If the application is a .NET assembly, the TI Automation proxy:

    a. Maps the COBOL data types to the .NET Framework data types.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

18. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

To implement this model, you must provide TI with an IP address, a port number, and a CICS program name to execute the application passed by the concurrent server program (Mscmtics.cbl).

Host Integration Server includes sample code showing how to implement the TCP ELM Link programming model. The sample code is located at \installation directory\SDK\Samples\AppInt. Start Microsoft Visual Studio, open the tutorial of your choice and follow the instructions in the Readme.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: CICS TCP/IP Socket Interface Guide (IBM Document #SC31-7131).

See Also

Tasks

Transaction Request Messages

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components

TI Runtime

Choosing the Appropriate Programming Model

Other Resources

Transaction Integrator Components

TCP Transaction Request Message User Data

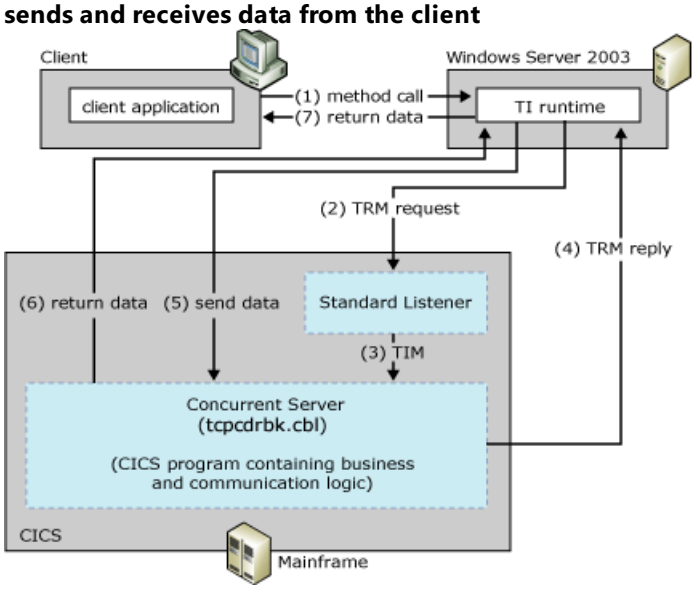
The TCP transaction request message (TRM) User Data programming model allows data and parameters to be exchanged directly between TI and the host TP. The TCP TRM User Data model is based on the CICS Concurrent Server model. The standard Listener uses two network exchanges to execute a single transaction program and requires the client to:

- Send a Transaction Request Message (TRM) to the standard Listener
- Receive a TRM reply from the application program
- Send the application request data stream to the server transaction program

Receive the application reply data from the server transaction program

The following figure summarizes the workflow occurring between the client, the standard CICS Listener, and the Concurrent Server. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Process by which the client starts the default Listener, which passes the call to the concurrent server, which then sends and receives data from the client



Summary Workflow Diagram for the TCP TRM User Data Programming Model

The TCP TRM User Data programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Reads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:

a. Calls the conversion routines to convert the application data to mainframe COBOL types.

b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. Passes the message to the TCP transport component.

4. The TI TCP transport sends a connect request to the standard Listener using the Internet Protocol (IP) address of the mainframe computer and the port address of the Listener.

5. The standard Listener accepts the connection request and tells TI runtime to send the TRM. The standard Listener then waits for the TRM.

   The TRM is a formatted data record that identifies the server TP to be invoked by using its TRANID. The CICS Listener TP is a special mainframe TP, whose main function is to receive server TP invocations sent by client applications running TCP/IP.

   The TRANID of the IBM-provided, standard Listener TP is CSKL. The TP name of the Listener TP as it appears in the program control table (PCT) is EZACIC02.

6. TI runtime formats the TRM and sends it to the standard Listener. TI waits for the TRM reply.

7. The standard Listener receives the TRM, sends TI runtime a receive confirmation, and then reads the contents of the TRM. The Listener interprets the information in the TRM and extracts the transaction ID of the Concurrent Server program that is to service the request.

8. The standard Listener starts the concurrent server TP program that is identified by the TRANID in the TRM (Mscmtics.cbl sample application) using EXEC CICS Start.

   Mscmtics.cbl is the Microsoft sample TP file that is used to pass data between COMTI and the server TP using the COMMAREA. The Mscmtics.cbl sample TP is developed by Microsoft and provided as part of the Host Integration Server software. It is located in the $\Microsoft Host Integration Server\SDK\Samples\Comti\ProgrammingSpecifics\Tcp. It must be compiled, linked, and installed on the mainframe computer prior to using this model.

Note

If the standard Listener is unable to start the Concurrent Server, the Listener formats an error message and sends it back to t he COMTI TCP Transport. Reasons the Listener might be unable to start include:

- rejected connection due to limited CICS resources (for example, exceeds the maximum number of CICS tasks or concurrent server tasks)
- invalid or disabled TRANID for the concurrent server
- invalid, disabled or unavailable Concurrent Server program associated with the transaction ID

Note

The error message from the CICS listener is character based and always begins with the letters EZY. The length of the error m essage is variable, and the end of the message is determined by the socket closed by the CICS Listener.

1. The standard Listener calls the socket application protocol interface (API) in the host environment. The standard Listener cannot send the TRM Reply. The TRM Reply represents a synchronization process that allows time for the transaction program to be started prior to the application request data being sent by the client. This synchronization process is necessary due to internal CICS architectural consideration (there is no guarantee as to when a transaction program is started after the request is made).

After the standard CICS Listener has issued the start command for the concurrent server transaction, the standard Listener is out of the application processing loop and is free to listen for another incoming request.

2. After the concurrent server is running, it reads the transaction initial message (TIM) sent by the standard Listener.

   The TIM describes the TCP/IP environment in which the server is running and contains the TCP/IP socket information the concurrent server uses to communicate with the COMTI TCP Transport and the client message header the concurrent server uses to customize its execution behavior.

3. The Concurrent Server:

   a. Formats the TRM reply.

   b. Sends a TRM Reply to the TI TCP Transport to inform it that it can now send the application request data.

   c. Issues a receive and waits for the application request data.

   Sending of the TRM Reply completes the 1st part of the Standard Listener exchange sequence.

4. TI evaluates the TRM and passes the data to the Concurrent Server. TI also sends socket shutdown, and then TI waits for the reply data.

5. After the Concurrent Server receives the application request data, the server performs the business logic on the data.

6. After the server has finished processing the request and formulating the reply, prepares the reply data and then sends the response directly to the client. Completing the processing of the application data signals the end of the 2nd exchange sequence.

7. The concurrent server closes the socket.

8. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

   a. Receives the message from the TCP transport component.

   b. Reads the message buffer.

   If the application is a COM+ component, the TI Automation proxy:

   a. Maps the COBOL data types to the automation data.

   b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

   If the application is a .NET Framework assembly, the TI Automation proxy:

   a. Maps the COBOL data types to the .NET Framework data types.

   b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

9. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

To implement this model, you must provide TI with an IP address, a port number, and a CICS program name to execute the application passed by the concurrent server program (Mscmtics.cbl). The model requires the installation, within CICS, of the IBM-supplied default Listener (EZACIC02). The CICS IBM default Listener uses IBM-provided default settings.

Host Integration Server includes sample code showing how to implement the TCP TRM Link programming model. The sample

code is located at *****installation directory***\SDK\Samples\AppInt**. Start Visual Studio, open either the tutorial you want to use, and follow the instructions in the **Readme**.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: CICS TCP/IP Socket Interface Guide (IBM Document #SC31-7131).

See Also

Tasks

Transaction Request Messages

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components

TI Runtime

Choosing the Appropriate Programming Model

Other Resources

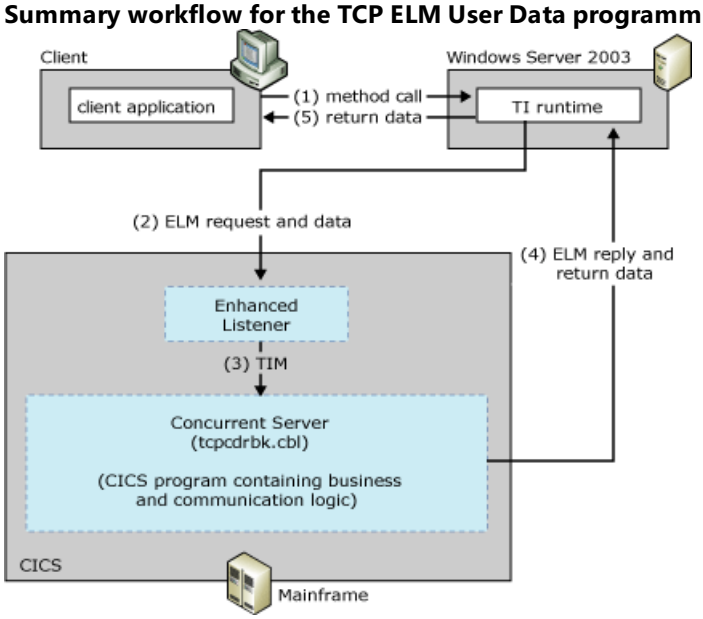
Transaction Integrator Components

TCP Enhanced Listener Message User Data

The TCP enhanced listener message (ELM) User Data model allows data and parameters to be passed directly between TI and the server TP.

The following figure summarizes the workflow occurring between the client, the enhanced CICS Listener, the Concurrent Server, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Summary workflow for the TCP ELM User Data programming model



TCP ELM User Data Programming Model

The TCP ELM User Data programming model works as follows:

1. An application invokes a method in a TI component configured in Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Reads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to mainframe COBOL types.
    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.
    c. Passes the message to the TCP transport component.
4. The TI TCP transport sends a connect request to the enhanced Listener using the Internet Protocol (IP) address of the mainframe computer and the port address of the Listener.

5. The enhanced Listener accepts the connection request and tells TI run-time to send the ELM. The enhanced Listener then waits for the ELM.

   The ELM is a formatted data record that identifies the server TP to be invoked by using its TRANID. The Listener TP is a special mainframe TP, whose main function is to receive server TP invocations sent by client applications running TCP/IP.

6. TI run-time formats the ELM and sends it to the enhanced Listener. TI then bypasses the transport logic that waits for a ELM reply and immediately sends the application request data after the request header. TI then waits for the ELM reply.

7. The enhanced Listener receives the 35 byte ELM, and then reads the contents of the ELM header. The enhanced Listener places the 35 bytes in the transaction initial message (TIM) but does not operate on its content.

   The TIM describes the TCP/IP environment in which the server is running and contains the TCP/IP socket information the concurrent server uses to communicate with the COMTI TCP Transport and the client message header the concurrent server uses to customize its execution behavior. The header contains the name of the server program to be linked to.

8. The enhanced Listener starts the concurrent server TP program (Mscmtics.cbl sample application) that is identified by the TRANID in the ELM using EXEC CICS Start.

   Mscmtics.cbl is the Microsoft sample TP file that is used to pass data between TI and the server TP using the COMMAREA. The Mscmtics.cbl sample TP is developed by Microsoft and provided as part of the Host Integration Server software. It is located in the $\Microsoft Host Integration Server\SDK\Samples\Comti\ProgrammingSpecifics\Tcp. It must be compiled, linked, and installed on the mainframe computer prior to using this model.

Note

If the standard Listener is unable to start the Concurrent Server, the Listener formats an error message and sends it back to t he COMTI TCP Transport. Reasons the Listener might be unable to start include:

- rejected connection due to limited CICS resources (for example, exceeds the maximum number of CICS tasks or concurrent server tasks)
- invalid or disabled TRANID for the concurrent server
- invalid, disabled or unavailable Concurrent Server program associated with the transaction ID

Note

The error message from the CICS listener is character based and always begins with the letters EZY. The length of the error m essage is variable, and the end of the message is determined by the socket closed by the CICS Listener. The enhanced Listene r calls the socket application protocol interface (API) in the host environment. After the enhanced Listener has issued the start command for the concurrent server transaction, the enhanced Listener is out of the application processing loop and is free to listen for another incoming request.

1. After the concurrent server is running, it reads the transaction initial message (TIM) sent by the standard Listener.

   The TIM describes the TCP/IP environment in which the server is running and contains the TCP/IP socket information the concurrent server uses to communicate with the COMTI TCP Transport and the client message header the concurrent server uses to customize its execution behavior.

2. The concurrent server sends the TRM to TI and waits for the application request data.

3. TI evaluates the TRM and passes the data directly to the concurrent server program (Mscmtics.cbl). TI also sends socket shutdown, and then TI waits for the reply data.

4. After the data is received, the server TP performs the business logic on the data. All business logic is defined in the server

TP.

5. The server TP prepares the reply data and then sends the response directly to the client.
6. The concurrent server closes the socket
7. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:
    a. receives the message from the TCP transport component.
    b. reads the message buffer

    If the application is a COM+ component, the TI Automation proxy:
    a. maps the COBOL data types to the automation data
    b. calls the conversion routines to convert the mainframe COBOL types to the application data

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. maps the COBOL data types to the .NET Framework data types
    b. calls the conversion routines to convert the mainframe COBOL types to the application data
8. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Host Integration Server includes sample code showing how to implement the TCP ELM User Data programming model. The sample code is located at \installation directory\SDK\Samples\AppInt. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the Readme.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: CICS TCP/IP Socket Interface Guide (IBM Document #SC31-7131).

See Also

Tasks

Transaction Request Messages

Reference

Configure Host Environment and Programming Model Wizard Page
Converting Data Types from Automation to OS/390 COBOL
Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components
TI Runtime
Choosing the Appropriate Programming Model

Other Resources

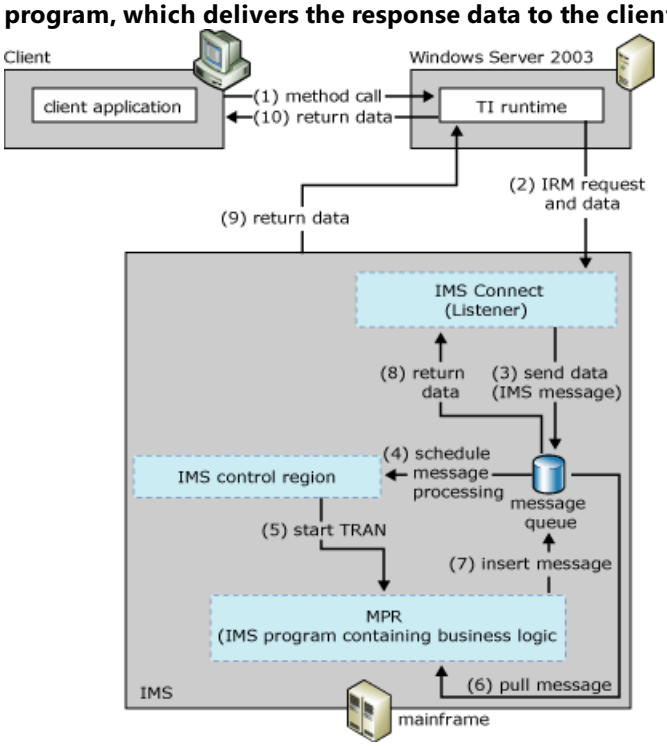
Transaction Integrator Components

IMS Connect

The IMS Connect programming model provides access to information management systems (IMS) transactions using TCP/IP. This model uses the IMS message queue for processing data.

The following figure summarizes the workflow occurring between the client, the default IMS Listener, the Concurrent Server, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Process by which the client passes input data to the ITOC listener and the HWSIMSO0 provides access to the IMS program, which delivers the response data to the client



Summary Workflow Diagram for the IMS Connect Programming Model

The IMS Connect programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Rreads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET data types to COBOL data types.

    The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to mainframe COBOL types.
    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.
    c. Passes the message to the TCP transport component.

4. The TI runtime sends an initial request message (IRM) to the IMS Connect, either HWSIMSO0 or HWSIMSO1, using the Internet Protocol (IP) address of the mainframe computer and the port address of the IMS Connect as stored in the TCP/IP profile data set (hlq.PROFILE.TCPIP) supplied by IBM.

   HWSIMSO0 and HWSIMSO1 are IBM-supplied host web server (HWS) exit routines that define the request and reply protocols between the TI Automation server (a TI component in a COM or .NET Framework application) and ITOC. The HWS runs in an MVS address space that is separate from the IMS regions and performs the listener services for the IMS connection.

5. The IMS Connect exit routine takes control of the IMS application (referred to as the IMS TCP/IP Open Transaction Management Architecture (OTMA) Connection (ITOC)).

6. The TI run-time environment sends an ITOC request header to ITOC and HWSIMSO0.

7. The HWSIMSO0 exit routine:

   - Validates the ITOC request header
   - Receives all request data from the TI run-time environment
   - Interfaces with security routines
   - Drives the OTMA process to connect to an IMS data store
   - Places and retrieves message segments into and from the IMS message queue through OTMA
   - Sends all reply data segments to the TI run-time environment
   - Controls recovery operations within IMS

8. ITOC reads the ITOC header information, locates the correct IMS region, and schedules the execution of an IMS transaction in that IMS region. The ITOC header must contain this information:

   - ITOC HWS exit routine identifier (default '*IRMREQ*')
   - IMS data store identifier
   - Transaction identifier
   - Flow control information
   - IBM's Resource Access Control Facility (RACF) security credentials
   - Protocol control flags

9. HWSIMSO0 schedules the correct IMS message queue

10. The TI run-time sends the request data segments to ITOC

11. The TI run-time sends EOM

12. IMS Control region sends to message processing region (MPR)

13. After all request data is placed on the IMS message queue, the transaction is scheduled for execution

14. The IMS server application program uses the standard CBLTDLI Get Unique (GU), Get Next (GN), and Insert (INSRT) call interface commands to retrieve the request data and to place reply data on the IMS message queue.

15. MPR returns data to TI. ITOC sends EOM-CSMOKY ITOC returns the following information to the TI run-time environment:

    - Request mod message
    - Reply data segments
    - End-of-message segment
    - CSMOKY segment

16. ITOC and the ITOC exit routine then remove the reply data from the message queue and deliver it back to the TI run-time environment.

17. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. receives the message from the TCP transport component.

    b. reads the message buffer

    If the application is a COM+ component, the TI Automation proxy:

    a. maps the COBOL data types to the automation data

    b. calls the conversion routines to convert the COBOL data types to the application data

    If the application is a .NET Framework assembly, the TI Automation proxy:

    a. maps the COBOL data types to the .NET Framework data types

    b. calls the conversion routines to convert the COBOL data types to the application data

18. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: IMS TCP/IP Application Developers Guide (IBM Document #SC31-7186) and IMS Connect Guide and Reference V1R2 (IBM Document #SC27-0946).

Host Integration Server includes sample code showing how to implement the IMS Connect programming model. The sample code is located at *****installation directory***\SDK\Samples\AppInt**. Start Visual Studio, open the tutorial you want to use, and follow the instructions in the **Readme**.

See Also

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

IMS Components

TI Runtime

Choosing the Appropriate Programming Model

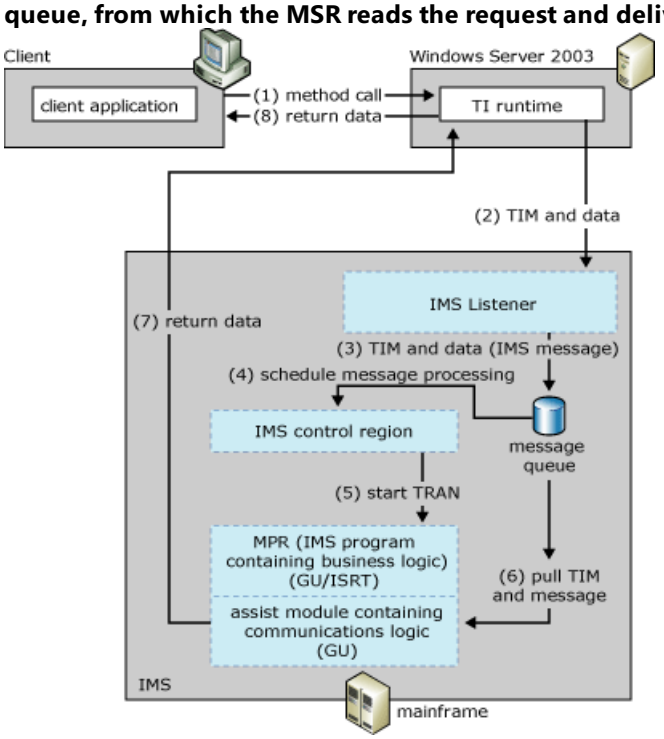
Other Resources

IMS Implicit

The IMS Implicit programming model provides access to information management systems (IMS) transactions using TCP/IP. This model uses the IMS message queue for processing data. This model is known as an implicit mode of communication because each IMS application implicitly issues all socket calls.

The following figure summarizes the workflow occurring between the client, the default IMS Listener, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Process by which the client passes input data to the BMP Listener, which in turn puts the data into the message queue, from which the MSR reads the request and delivers the response data to the client



Summary workflow diagram for the IMS Implicit programming model

The IMS Implicit programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Reads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to mainframe COBOL types.
    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. Passes the message to the TCP transport component.

4. The TI Automation proxy sends a TRM and the application data to the default IMS Listener using the proper Internet Protocol (IP) address of the mainframe computer and the port address of the default Listener.

   The TRM is a formatted data record that identifies the IMS server TP to be invoked by using its TRANID. EZAIMSLN is the default Listener TP supplied by IBM, whose main function is to receive server TP invocations sent by client applications running TCP/IP.

5. The Listener receives the TRM and data and then creates a transaction initiation message (TIM)

6. The Listener inserts the TRANID along with the client data into the IMS message queue that is associated with the MPP region in which the server TP is running. The server TP runs concurrently with the Listener.

7. After the connection is established, the Listener releases control and continues to listen for more client TCP/IP calls. The Listener is able to maintain multiple concurrent connections.

8. The IMS control region schedules the transaction in a message processing region (MPR) and passes the socket to the IMS server TP.

9. The server TP is coded to use the IBM-supplied Assist Module (CBLADLI, instead of the standard COBOL DL/I module, CBLTDLI) to intercedes between the server TP and the TI run-time environment, thereby allowing the server TP to exchange data with TI by using DL/I calls (GU, GN, and ISRT).

   The Assist Module translates IMS programming calls to TCP/IP socket API calls that are understood by TCP/IP client applications such as TI. When developing IMS TPs, you do not need to learn the TCP/IP sockets interface because the Assist Module performs all of the necessary translations.

10. The server TP calls the Assist Module and reads the request from the queue using GU and GN commands. All business logic is defined in the concurrent server TP.

11. After processing of the data is complete, the server TP uses the DL/I ISRT call to return data to the TI runtime by way of TCP/IP.

12. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the TCP transport component.

    b. Reads the message buffer.

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the COBOL data types to the automation data

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data

    If the application is a .NET Framework assembly, the TI Automation proxy:

    a. Maps the COBOL data types to the .NET Framework data types.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

13. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Host applications are written using standard IMS commands. However, the commands use the CBLADLI application programming interface (API), rather than the standard CBLTDLI API.

In addition, the proper IMS control regions must be defined in an APPL statement in VTAM.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: IMS TCP/IP Application Developers Guide (IBM Document #SC31-7186).

Host Integration Server includes sample code showing how to implement the IMS Connect programming model. The sample code is located at \installation directory\SDK\Samples\AppInt. Start Visual Studio, open the tutorial you want to use, and follow the instructions in the Readme.

See Also

Reference

Configure Host Environment and Programming Model Wizard Page
Converting Data Types from Automation to OS/390 COBOL
Converting Data Types from OS/390 COBOL to Automation

Concepts

IMS Components
TI Runtime
Choosing the Appropriate Programming Model

Other Resources

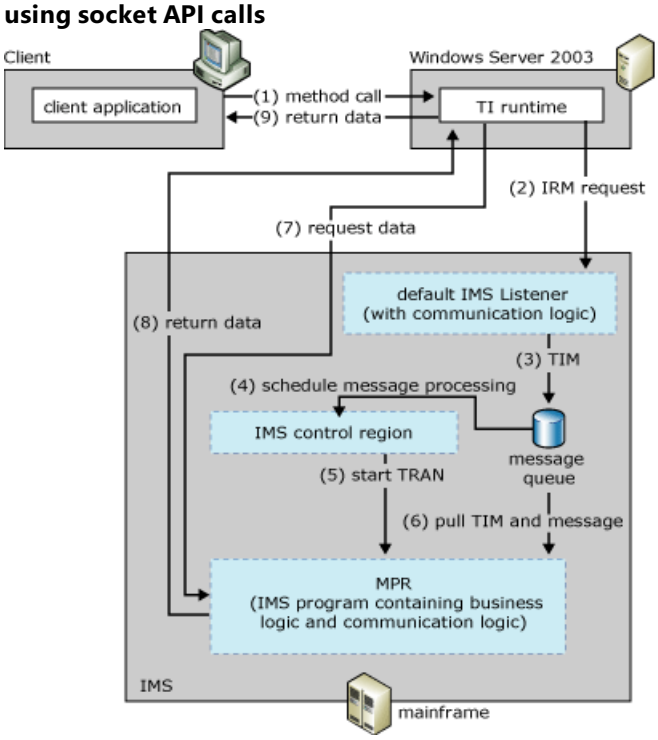
Transaction Integrator Components

IMS Explicit

The IMS Explicit programming model provides access to IMS transactions by using TCP/IP. This model does not use the IMS message queue for processing data. This model is known as an explicit mode of communication because each IMS application must specify all socket calls explicitly.

The following figure summarizes the workflow occurring between the client, the default IMS Listener, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Process by which the client passes input data to the BMP listener, which in turn puts the data into the message queue, from which the IMS region schedules execution in the MPR and delivers the response data to the client using socket API calls



Summary Workflow Diagram for the IMS Explicit Programming Model

The IMS Explicit programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Reads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to mainframe COBOL types.
    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. Passes the message to the TCP transport component.

4. The TI Automation proxy sends a TRM and the application data to the default IMS Listener using the proper Internet Protocol (IP) address of the mainframe computer and the port address of the default Listener.

   The TRM is a formatted data record that identifies the IMS server TP to be invoked by using its TRANID. EZAIMSLN is the IBM-supplied default Listener TP, whose main function is to receive server TP invocations sent by client applications running TCP/IP.

5. The Listener receives the TRM and creates a transaction initiation message (TIM).

6. The Listener inserts the TRANID along with the client data into the IMS message queue that is associated with the message processing region (MPR) in which the server TP is running. The server TP runs concurrently with the Listener.

Note
All IMS host server programs must be administered to IMS as NonResponse transactions.

7. After the connection is established, the Listener releases control and continues to listen for more client TCP/IP calls. The Listener is able to maintain multiple concurrent connections.

8. The IMS control region schedules the transaction in the MPR and passes the socket to the IMS server TP.

9. The server TP uses the standard COBOL DL/I module, CBLTDLI, to initiate the transfer of data from the TI run-time environment by way of APPC/MVS.

   The server TP uses a series of socket API calls to send and receive data directly with the TI runtime. You will need to learn the TCP/IP sockets interfaces and make those calls explicitly in your program.

10. After processing of the data is complete, the server TP uses the DL/I ISRT call to return data to the TI runtime by way of APPC/MVS.

11. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the TCP transport component.

    b. Reads the message buffer.

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the COBOL data types to the automation data.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

    If the application is a .NET assembly, the TI Automation proxy:

    a. Maps the COBOL data types to the .NET Framework data types.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

12. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

In addition, the proper IMS control regions must be defined in an APPL statement in VTAM.

Host Integration Server includes sample code showing how to implement the IMS Connect programming model. The sample

code is located at *****installation directory***\SDK\Samples\AppInt**. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the **Readme**.

For information about configuring the mainframe and writing server applications for TCP/IP, see TCP/IP V3R2 for MVS: IMS TCP/IP Application Developers Guide (IBM document #SC31-7186).

See Also

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

IMS Components

TI Runtime

Choosing the Appropriate Programming Model

Other Resources

Transaction Integrator Components

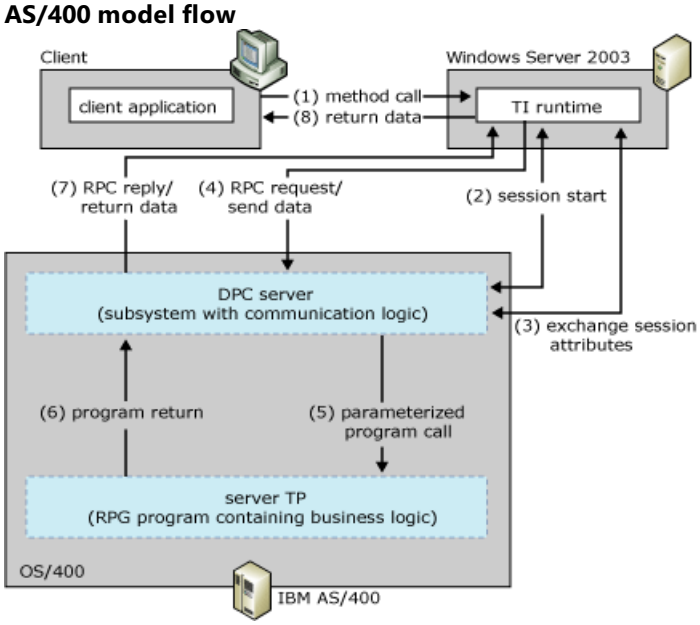
OS/400 Distributed Program Calls

The OS/400 Remote Command and Distributed Program Calls (DPC) programming model allows most AS/400 applications to interact with TI in request-reply fashion (client-initiated only) with minimum modifications. DPC is a documented protocol that supports program to program integration on an AS/400, which can be accessed easily from PC based applications using the TCP/IP networking protocol.

Note
This interface does not support host-initiated processing (HIP); AS/400 integration is for client-initiated calls only.

The following figure summarizes the workflow occurring between the client, the default DPC Server, and the AS/400 transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

AS/400 model flow



Summary Workflow Diagram for the OS/400 DPC Programming Model

The OS/400 DPC programming model works as follows:

1. An application invokes a method in a TI component configured in Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is COM+ component, the TI Automation proxy:
    a. Reads in the type libraries previously created by the TI Designer.
    b. Maps the automation data types to AS/400 RPG data types.

    If the application is a .NET Framework assembly, the TI Automation proxy:
    a. Reads in the assembly and metadata previously created by the TI Designer.
    b. Maps the .NET Framework data types to AS/400 RPG data types.

    The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to AS/400 RPG types.
    b. Builds the parameterized message buffer that represents the RPG PLIST.

c. Passes the message to the AS/400 DPC transport component.

4. The TI TCP transport sends a connection request to the DPC Server system using the Internet Protocol (IP) address of the AS/400 computer and the port address of the server. The TI TCP transport then waits for a reply.

5. The DPC Server on the AS/400 accepts the session request and issues a receive. The DPC Server then waits for the start server request.

6. The TI automation proxy sends the DPC Server a start server request and issues a receive. The TI TCP transport then waits for a start server reply.

7. The DPC server processes the start server request, sends a start server reply, and then issues a receive. The DPC Server then waits for an exchange attributes request.

8. The TI runtime processes the start server reply, sends the attributes request, and issues a receive. The TI runtime then waits for an exchange attributes reply.

9. The DPC server processes the exchange attributes request, sends a exchange attributes reply, and then issues a receive. The DPC then waits for a remote program call request.

10. TI runtime processes the exchange attributes reply and then sends remote program call request followed immediately by remote program call reply and the converted data.

11. The DPC server processes the request, sends remote program call reply followed by remote program call parameters and data.

12. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the TCP transport component.

    b. Reads the message buffer.

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the OS/400 data types to the automation data.

    b. Calls the conversion routines to convert the OS/400 RPG types to the application data.

    If the application is a .NET assembly, the TI Automation proxy:

    a. Maps the AS/400 data types to the .NET Framework data types.

    b. Calls the conversion routines to convert the OS/400 RPG types to the application data.

13. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Note
The maximum size of a message is 32,767 bytes, including field headers and data.

Note
The RMTPGMCALL can pass maximum 35 parameters as IN or OUT, or as IN/OUT in any combination.

Host Integration Server includes sample code showing how to implement the IMS Connect programming model. The sample

code is located at *****installation directory***\SDK\Samples\AppInt**. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the **Readme**.

For information about configuring the mainframe and writing server applications for IBM AS/400e, see the ILE RPG/400 Programmers Guide Version 4 (IBM Document #SC09-2507-02) and the ILE RPG/400 Reference Version 3 (IBM Document #SC09-2077-01).

See Also

Reference

Configure Host Environment and Programming Model Wizard Page
Converting Data Types from RPG to Automation
Converting Data Types from Automation to RPG

Concepts

AS/400 Security
TI Runtime
Choosing the Appropriate Programming Model

Other Resources

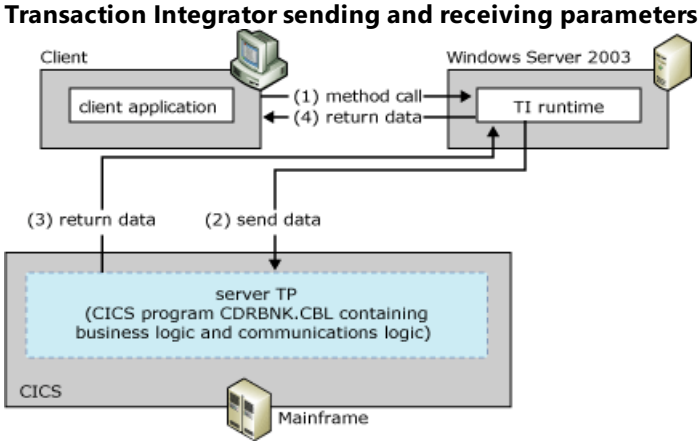
Transaction Integrator Components
COMTIContext Interface

CICS LU6.2 Link

The CICS LU6.2 Link programming model is one of the simplest models that you can use to implement TI functionality.

The following figure summarizes the workflow occurring between the client, the default CICS Mirror Transaction, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Transaction Integrator sending and receiving parameters with DPL information from a CICS Mirror Transaction



Summary workflow diagram for the CICS LU6.2 Link programming model

The CICS LU6.2 Link programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.

2. The TI runtime calls the TI Automation proxy.

3. If the application is a COM+ component, the TI Automation proxy does the following:

    a. Reads in the type library created previously by the TI Designer.

    b. Maps the automation data types to COBOL data types.

    If the application is a .NET assembly, the TI Automation proxy does the following:

    a. Reads in the assembly and metadata created previously by the TI Designer.

    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:

    a. Calls the conversion routines to convert the application data to mainframe COBOL types.

    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

    c. Passes the message to the SNA transport component.

4. TI sends the TP Name CSMI request specified by the TI component method to the CICS Mirror Transaction using DPL information and the LU6.2 protocol. (IBM provides CSMI with CICS on the TI prerequisite systems.)

    The CICS Mirror Transaction is a special CICS TP that acts as a gateway between TPs running in different CICS regions, thereby allowing them to exchange data through the COMMAREA. TI takes advantage of this standard method of communication between CICS TPs to access mainframe TPs. CSMI handles all APPC and transactional properties required on the communication. The TRANID for this TP is CSMI.

    The Distributed Program Link (DPL) is the protocol used when communicating with CSMI. TI uses DPL to communicate

with CSMI.

5. CSMI (the CICS Mirror Transaction) takes control and issues an EXEC CICS Link command to the requested server TP in CICS. (The name of this program can be associated the remote environment (RE) and with the method name in TI Designer.)

6. The CICS Mirror transaction passes the COMMAREA that contains the input fields to the server TP.

   The COMMAREA is a communication area of up to 32 KB containing all of the data that is passed to and from the mainframe program. Many CICS TPs, written in COBOL, use this area of the mainframe transaction code to exchange data. When using the CICS Link using LU6.2 programming model, TI appears to the mainframe TP as just another CICS TP exchanging data through the COMMAREA.

   The Server TP is the TP that TI invokes on behalf of the client application. It contains the business logic being executed and is identified by its TRANID in the method call of the client application.

Note
The term server TP is used to identify the TP that TI is accessing. This clarification is necessary because access to mainframe applications may and typically does involve a number of TPs.

7. When the server TP is finished processing, it issues an EXEC CICS RETURN command, which returns the data in the COMMAREA to the CICS Mirror transaction with all output fields updated.

8. The CICS Mirror transaction returns the output data, if any is required, to TI.

9. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

   a. receives the message from the SNA transport component.

   b. reads the message buffer

   If the application is a COM+ component, the TI Automation proxy:

   a. maps the COBOL data types to the automation data

   b. calls the conversion routines to convert the mainframe COBOL types to the application data

   If the application is a .NET assembly, the TI Automation proxy:

   a. maps the COBOL data types to the .NET Framework data types

   b. calls the conversion routines to convert the mainframe COBOL types to the application data

10. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Only the flow model is supported with CICS Link, so unbounded recordsets are not supported for this class of TP. Fixed-sized recordsets (that is, bounded recordsets) are supported.

CSMI also handles any Sync Level 2 interactions with TI, and thus transparently provides the 2PC capability for programs in this class.

Existing CICS programs may already be structured this way. Instead of TI issuing the LU 6.2 request, another CICS TP might already issue an EXEC CICS Link to run the CICS program shown in the previous illustration. In that case, both the existing CICS TP and the TI component can coexist and run the same CICS program.

Note

CSMI is the default mirror transaction name, but you can specify a different name.

Host Integration Server includes sample code showing how to implement the CICS LU6.2 Link programming model. The sample code is located at \installation directory\SDK\Samples\AppInt. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the Readme.

See Also

Reference

Configure Host Environment and Programming Model Wizard Page
Converting Data Types from Automation to OS/390 COBOL
Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components
TI Runtime
Choosing the Appropriate Programming Model

Other Resources

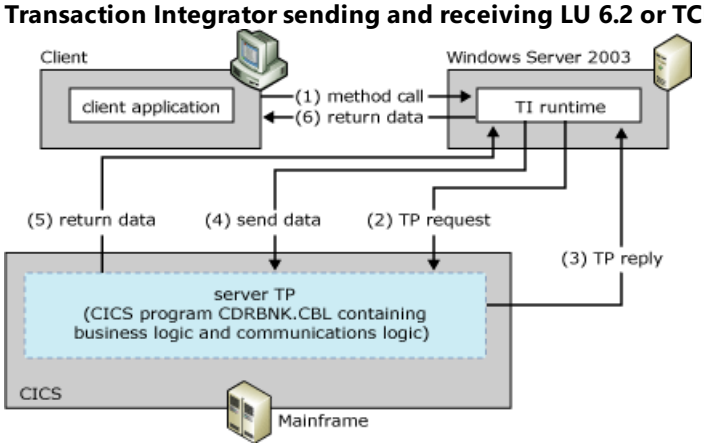
Transaction Integrator Components

CICS LU6.2 User Data

The CICS LU6.2 User Data programming model provides direct invocations and data exchanges between TI and the server TP. No other communication components are required with this model.

The following figure summarizes the workflow occurring between the client, the default CICS Mirror Transaction, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Transaction Integrator sending and receiving LU 6.2 or TCP/IP from the mainframe transaction program



Summary workflow diagram for the CICS LU6.2 User Data programming model

The CICS LU6.2 User Data programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.

2. The TI runtime calls the TI Automation proxy.

3. If the application is a COM+ component, the TI Automation proxy:

    a. Reads in the type library created previously by the TI Designer.

    b. Maps the automation data types to COBOL data types.

    If the application is a .NET assembly, the TI Automation proxy:

    a. Reads in the assembly and metadata created previously by the TI Designer.

    b. Maps the .NET Framework data types to COBOL data types.

    The TI Automation proxy then:

    a. Calls the conversion routines to convert the application data to mainframe COBOL types.

    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

    c. Passes the message to the SNA transport component.

4. The TI proxy sends the TP invocation request specified by the TI component method to the server TP by using the LU6.2 protocol. In this message, TI sends the TRANID of the server TP that the method is invoking.

5. TI and the server TP communicate directly by issuing APPC or Common Programming Interface for Communications (CPI-C) verbs to receive and send the input and output fields, respectively.

6. If necessary, the server TP issues the appropriate verbs to implement Sync Level 2 properties and 2 phase commit.

7. The mainframe TP closes the socket.

8. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

    a. Receives the message from the SNA transport component.

    b. Reads the message buffer

    If the application is a COM+ component, the TI Automation proxy:

    a. Maps the COBOL data types to the automation data.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

    If the application is a .NET assembly, the TI Automation proxy:

    a. Maps the COBOL data types to the .NET Framework data types.

    b. Calls the conversion routines to convert the mainframe COBOL types to the application data.

9. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Host Integration Server includes sample code showing how to implement the CICS LU6.2 User Data programming model. The sample code is located at *****installation directory***\SDK\Samples\AppInt**. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the **Readme**.

See Also

Reference

Configure Host Environment and Programming Model Wizard Page

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Concepts

CICS Components

TI Runtime

Choosing the Appropriate Programming Model

Other Resources

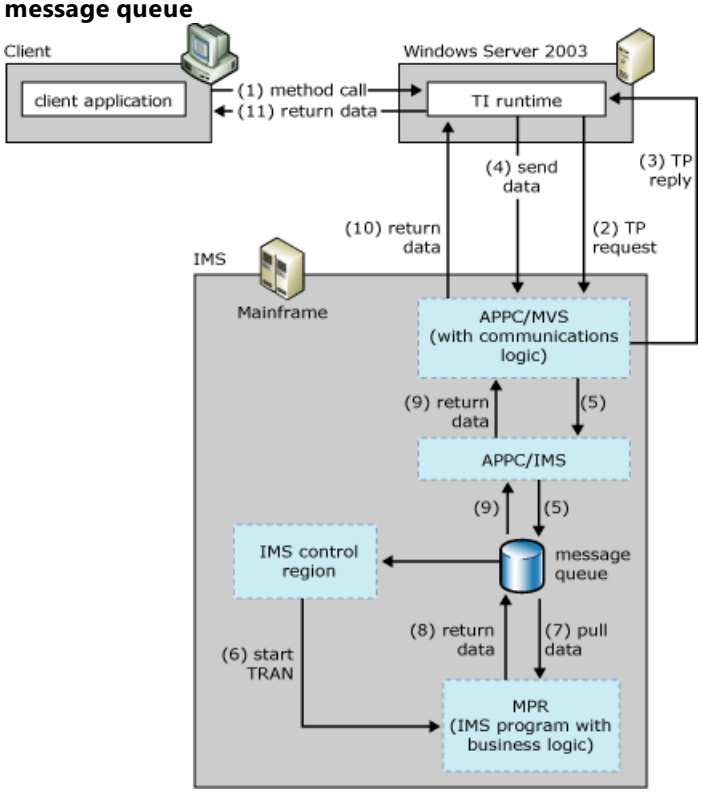
Transaction Integrator Components

IMS LU6.2 User Data

The IMS LU6.2 programming model provides access to IMS transactions using LU6.2.

The following figure summarizes the workflow occurring between the client, the default IMS Listener, and the mainframe transaction program. The numbers in parentheses indicate the approximate order in which events occur. A more detailed description of the events follows the figure.

Transaction Integrator sending and receiving LU 6.2 from MVS/APPC, which then sends and receives from the IMS message queue



Summary workflow diagram for the IMS LU6.2 User Data programming model

The IMS LU6.2 programming model works as follows:

1. An application invokes a method in a TI component configured in either Component Services or the .NET Framework.
2. The TI runtime calls the TI Automation proxy.
3. If the application is a COM+ component, the TI Automation proxy:
    a. Reads in the type library created previously by the TI Designer.
    b. Maps the automation data types to COBOL data types.

    If the application is a .NET assembly, the TI Automation proxy:
    a. Reads in the assembly and meta data created previously by the TI Designer.
    b. Maps the .NET Framework data types to COBOL data types.
4. The TI Automation proxy then:
    a. Calls the conversion routines to convert the application data to mainframe COBOL types.
    b. Builds the flattened data stream buffer that represents the COBOL declaration or copybook.

c. Passes the message to the SNA transport component.

5. The TI Automation proxy sends the transaction execution request (TER) and the user data to MVS APPC through the IBM-supplied multiple virtual storage/advanced program-to-program communications (APPC/MVS) application.

6. APPC/MVS application instructs IMS to place the transaction execution request and user data on the IMS message queue.

7. IMS schedules the server TP into a message processing region (MPR).

8. After execution begins, the TP issues a DL/I Get Unique (GU) command to get the input parameters that were sent by the TI runtime. If there is an input unbounded record set, the TP also makes one or more Get Next (GN) calls to get each row of the record set that was sent.

9. After the TP processes the inputs and makes any database calls, it makes one or more Insert (ISRT) calls to place the output parameters and possibly an output or return-value unbounded recordset into the IMS message queue to be packaged and returned to the TI runtime through the APPC/MVS application.

10. The TI Automation proxy receives the reply data and processes the reply. The TI Automation proxy:

11. receives the message from the SNA transport component.

12. reads the message buffer

    If the application is a COM+ component, the TI Automation proxy:

13. maps the COBOL data types to the automation data types

14. calls the conversion routines to convert the mainframe COBOL types to the application data

    If the application is a .NET assembly, the TI Automation proxy:

15. maps the COBOL data types to the .NET Framework data types

16. calls the conversion routines to convert the mainframe COBOL types to the application data

17. The TI runtime sends the converted data back to the COM or .NET Framework application that invoked the method.

Host Integration Server includes sample code showing how to implement the IMS LU6.2 User Data programming model. The sample code is located at *****installation directory***\SDK\Samples\AppInt**. Start Microsoft Visual Studio, open the tutorial you want to use, and follow the instructions in the **Readme**.

See Also

Reference

Configure Host Environment and Programming Model Wizard Page
Converting Data Types from Automation to OS/390 COBOL
Converting Data Types from OS/390 COBOL to Automation

Concepts

IMS Components
TI Runtime
Choosing the Appropriate Programming Model

Other Resources

Transaction Integrator Components

Choosing the Appropriate Programming Model

A TI programming model determines the method used to access and integrate host applications and TI configuration requirements depending on the specific TI programming model being used. Implementing TI may require modification to the existing mainframe TPs to be able to fit the programming models that it supports. Specifically, this may be necessary when:

- A TP does not expect a simple request-reply response.
- A CICS TP has terminal processing logic embedded in the same TP with the business logic. This type of TP must be restructured as two separate TPs. Accesses business logic that already exists on the mainframe computer as TPs. You can use this function, or you can create the methods on the COM side and then create the necessary server TPs on the mainframe computer. This is still a viable option because TI may be better for accessing some types of data, such as those stored in VSAM data sets, than standard data access methods.

You must carefully analyze the business requirements of your organization so that you can implement transaction access by using one of the programming models provided in TI.

TI supports the programming models listed in the table below. Some of the factors you should consider when choosing the appropriate programming model for your organization are:

- the network procotol
- the maximum size of the message or data that can be sent to the host
- whether you need to use two-phase commit transactions in host applications
- whether you have to write your own communications protocol to support a Link program
- whether you want the server to have the ability to maintain the client to server context, also referred to as a persistent connection
- other requirements specific to a particular model.

The following table summarizes the similarities and differences among the programming models.

Programming Model	Network Protocol	Maximum Message or Data Size	Supports Two-phase Commit?	Write Own Communi-cations Protocol?	Supports Persistent Connect-ions?	Other Requirements
TCP Transaction Request Message Link	TCP/IP	32 KB	No	No (see sample code)	Yes	• See mscmtics.cbl sample application. • 1:many relationship between server application and port.
TCP Enhanced Listener Message Link	TCP/IP	32 KB	No	No (see sample code)	Yes	• See mscmtics.cbl sample application. • 1:1 relationship between server application and port.

TCP Transaction Request Message User Data	TCP/IP	unlimited	No	Yes (Server TPs are coded to handle all socket calls over TCP/IP)	Yes	• 1:many relationship between server application and port.
TCP Enhanced Listener Message User Data	TCP/IP	unlimited	No	Yes (Server TPs are coded to handle all socket calls over TCP/IP)	Yes	• 1:1 relationship between server application and port.
IMS Connect	TCP/IP	10MB	No	No	No	• No inbound (from TI to the host) unbounded recordsets are allowed. TI cannot send unbounded recordsets to the host; only those recordsets coming back from the host to TI are supported. • Dependent on the IBM supplied HWSIMSO0 and HWSIMSO0 exit routines.
IMS Implicit	TCP/IP	unlimited inbound 32KB outbound	No	No	No	
IMS Explicit	TCP/IP	unlimited	No	Yes (IMS TPs are coded to handle all socket calls over TCP/IP)	No	
OS/400 Distributed Program Calls	TCP/IP	32KB	No	No	Yes	

CICS LU6.2 Link	LU 6.2	32KB	Yes	No	No	• Server TPs are already coded to use the COMMAREA. Note CICS Link does not support multiple send-and-receive commands. Therefore, variable length record sets are not supported, but fixed-sized recordsets are supported. • CICS TPs do not contain the necessary logic to handle issuing APPC verbs directly, but instead must rely on the CICS Mirror transaction. • The TP is coded for a simple send-and-receive sequence.
CICS LU6.2 User Data	LU 6.2	unlimited	Yes	Yes (Server TPs are coded to handle all APPC and Sync Level 2 communica-tions.)	Yes	• Existing TPs contain the proper code necessary to manage their own APPC and Sync Level 2 communications. • Can use multiple send-and-receive commands.
IMS LU6.2 User Data	LU 6.2	unlimited	Yes	No	No	• Each server TP must have the embedded code necessary to handle all data communications using the LU6.2 protocol.

Implementing a specific programming model requires that you install and configure the appropriate software on your mainframe or AS/400 computer. When choosing the appropriate programming model for your organization, you might want assess how closely your current host configuration match the minimum requirements. The following table summarizes the minimum software and configuration requirements for each programming model.

Programming Model	**To use this model, you must install and configure**
TCP Transaction Request Message Link	• IBM MVS operating system 4.3 or later. • IBM CICS 4.0 or later. • The Listener TP, which is included in CICS TCP/IP, configured and started. • TCP/IP for MVS version 3.2 or later. • At least one CICS region defined in an APPL statement in VTAM with TPs configured.

TCP Enhanced Listener Message Link	• IBM MVS operating system 4.3 or later. • IBM CICS Transaction Server 1.3 or 2.0. • The Listener TP, which is included in CICS TCP/IP, configured and started. • TCP/IP for MVS version 3.2 or later. • At least one CICS region defined in an APPL statement in VTAM with TPs configured.
TCP Transaction Request Message User Data	• IBM MVS operating system 4.3 or later. • IBM CICS 4.0 or later. • The Listener TP, which is included in CICS TCP/IP, configured and started. • TCP/IP for MVS version 3.2 or later. • At least one CICS region defined in an APPL statement in VTAM with TPs configured.
TCP Enhanced Listener Message User Data	• IBM MVS operating system 4.3 or later. • IBM CICS Transaction Server 1.3 or 2.0. • The Listener TP, which is included in CICS TCP/IP, configured and started. • TCP/IP for MVS version 3.2 or later. • At least one CICS region defined in an APPL statement in VTAM with TPs configured.
IMS Connect	• IBM MVS operating system 4.3 or later. • IBM IMS 4.1 or later. • The Listener TP included in IMS TCP/IP. • TCP/IP for MVS version 3.2 or later. • IMS TCP/IP.

IMS Implicit	• IBM MVS operating system 4.3 or later. • IBM IMS 4.1 or later. • The Listener TP included in IMS TCP/IP. • TCP/IP for MVS version 3.2 or later. • IMS TCP/IP.
IMS Explicit	• IBM MVS operating system 4.3 or later. • IBM IMS version 4.1 or later. • The Listener TP included in IMS TCP/IP. • TCP/IP for MVS version 3.2 or later. • IMS TCP/IP.
OS/400 Distributed Program Calls	• IBM OS/400 release 4 version 1 or later.
CICS LU6.2 Link	• IBM MVS operating system version 4.3 or later. • IBM CICS version 3.3 or later. • The CICS Mirror transaction, which is included in CICS version 3.3 or later. • VTAM. • At least one CICS region defined in an Application (APPL) statement in VTAM with TPs configured. • The proper VTAM PU, LU, and Mode definitions necessary to establish Systems Network Architecture (SNA) connectivity
CICS LU6.2 User Data	• IBM MVS operating system 4.3 or later, including OS/390. • IBM CICS 3.3 or later. • VTAM. • At least one CICS region defined in an APPL statement in VTAM with TPs configured. • The proper VTAM PU, LU, and Mode definitions necessary to establish SNA connectivity.

IMS LU6.2 User Data	• IBM MVS operating system 4.3 or later. • MVS/APPC must be installed on the mainframe computer. MVS/APPC is included with the operating system. • IBM IMS 4.0 or later. • IBM IMS 6.0 or later if using 2PC protocols (Sync Point level 2). • IBM Recovery Resource Services (RRS) if using 2PC protocols (Sync Point level 2). In addition, the proper IMS control regions must be defined in an APPL statement in VTAM.

See Also

Concepts

Two-Phase Commit

Other Resources

Programming Models

Supported Data Flow Models

Transaction Integrator (TI) supports the following four data flows:

- Non-transactional Data Flows That Support Bounded Recordsets
- Non-transactional Data Flows That Support Unbounded Recordsets
- Transactional Data Flows That Support Bounded Recordsets
- Transactional Data Flows That Support Unbounded Recordsets

The following table shows the mainframe-based programming models and the data flows that TI supports.

Mainframe Programming Models	Non-transactional bounded recordsets	Non-transactional unbounded recordsets	Transactional bounded recordsets	Transactional unbounded recordsets
TCP Transaction Request Message (TRM) Link	X			
TCP Enhanced Listener Message (ELM) Link	X			
TCP Transaction Request Message (TRM) User Data	X	X		
TCP Enhanced Listener Message (ELM) User Data	X	X		
IMS Connect	X			
IMS Implicit	X			
IMS Explicit	X	X		
OS/400 Distributed Program Calls	X			
CICS LU6.2 Link	X		X	
CICS LU6.2 User Data	X	X	X	X
IMS LU6.2	X	X	X	X

In This Section

Non-transactional Data Flows That Support Bounded Recordsets

Non-transactional Data Flows That Support Unbounded Recordsets

Transactional Data Flows That Support Bounded Recordsets

Transactional Data Flows That Support Unbounded Recordsets

See Also

Other Resources

Programming Models

Non-transactional Data Flows That Support Bounded Recordsets

For each method invocation, TI converts and sends the input parameters to the transaction program (TP). The mainframe TP executes, processes the input data (for instance, accessing or updating the database), and sends its response back to TI. Then TI receives the output parameters from the TP and converts them to return to the invoker.

This data flow model does not support unbounded recordsets. (An unbounded recordset has no set number of rows.)

The following server models support this data flow model:

- CICS LU6.2 Link
- CICS LU6.2 User Data
- IMS LU6.2
- TCP Transaction Request Message (TRM) Link
- TCP Transaction Request Message (TRM) User Data
- IMS Explicit
- IMS Implicit
- IMS Connect
- OS/400 Distributed Program Calls

See Also

Other Resources

Supported Data Flow Models

Non-transactional Data Flows That Support Unbounded Recordsets

This model supports one or more consecutive sends followed by one or more consecutive receives. Therefore, the last input parameter of the method invocation can be an unbounded recordset (that is, a recordset with no set number of rows).

The input parameters are converted and sent, and then each row of the recordset is converted and sent to the mainframe transaction program (TP). The mainframe TP executes, receives the input data and each row of the recordset, processes the data (perhaps doing database accesses and/or updates), and then sends its response back to TI. TI receives any output parameters from the TP and converts them to return to the invoker. Then each row of an unbounded recordset is received, if appropriate. The mainframe TP has no notion of a recordset; it is just receiving or sending tabular data. TI handles all conversion to and from the recordset.

The following server models support this data flow model:

- CICS LU6.2 User Data
- IMS LU6.2
- TCP Transaction Request Message (TRM) User Data
- IMS Explicit

See Also

Other Resources

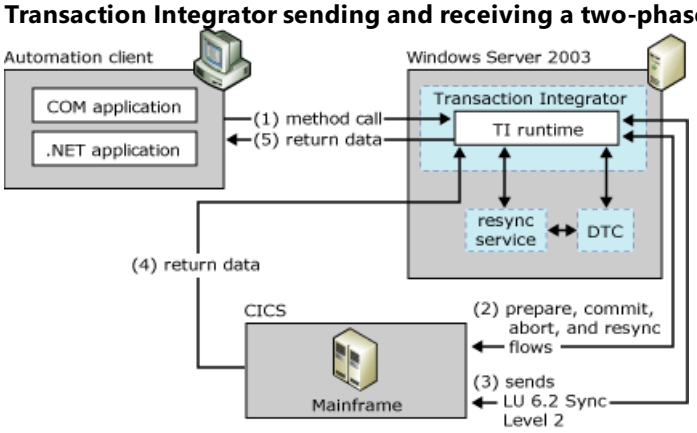
Supported Data Flow Models

Transactional Data Flows That Support Bounded Recordsets

This data flow model does not support unbounded recordsets. (An unbounded recordset has no set number of rows.)

The following figure shows what happens when a TI component participates in a COM+ transaction (a DTC-coordinated two-phase commit [2PC] exchange). TI uses LU 6.2 Sync Level 2 to manage the transaction. This capability applies to CICS and to IMS version 6.0 with Resource Recovery Services (RRS).

Transaction Integrator sending and receiving a two-phase commit exchange from a mainframe via LU 6.2



TI sending and receiving a two-phase commit

This transactional model does not support unbounded recordsets. (An unbounded recordset has no set maximum number of rows.) The transactional model is supported only by the LU 6.2 protocol, not the TCP/IP protocol. Therefore this model supports the following server models only:

- CICS LU6.2 Link
- CICS LU6.2 User Data
- IMS LU6.2

Microsoft Distributed Transaction Coordinator (DTC) works with the TI run-time environment and with the SNA LU 6.2 Resync TP service to provide the necessary 2PC flows as well as transaction log synchronization and recovery services. TI Automation clients can remain completely uninvolved in transaction commit or rollback decisions, or they can participate as shown in the following Visual Basic code. TI Automation clients are never aware of, nor involved in, transaction recovery. The following code sample is for a Visual Basic client that is involved in transaction control.

```
'Get object context reference
Set ctxObject = GetObjectContext()
'Create object instance for an example ProgID
Set obj = ctxObject.CreateInstance ("A.B.1")
'Invoke a method on the object with some parameters
ret = obj.YourMethod(c,d,e)
Do any other work that is part of this transaction
'If application finds all is well, then commit, otherwise roll back
If<something is not OK> Then
   ctxObject.SetAbort
Else
   ctxObject.SetComplete
End if
```

If a client remains uninvolved in transaction control, the TI run-time environment automatically commits transactions that have no run-time environment failures and automatically rolls back those transactions that have failures. The TI run-time environment, however, cannot detect or react to application-specific conditions that require a transaction to initiate a rollback. These must be handled by the client application. For example, an "out of cash" situation in an automated teller machine must be handled by the client application.

See Also

Other Resources

Supported Data Flow Models

Transactional Data Flows That Support Unbounded Recordsets

This model supports one or more consecutive sends followed by one or more consecutive receives. Therefore, the last input parameter and/or the return value of the method invocation can be an unbounded recordset (that is, a recordset with no set number of rows).

The CICS Link programming model cannot be used, so this data flow model supports only these two server models:

- CICS LU6.2 User Data
- IMS LU6.2

See Also
Other Resources
Supported Data Flow Models

Iterative vs. Concurrent TCP/IP Models

IBM defines two models for accessing server applications in CICS and IMS. In both models, there is a TCP/IP connection Listener and a Server aspect to the total application. The manner in which the Listener and Server portions of the application are implemented determines whether the model is Iterative or Concurrent.

- Iterative model. In the Iterative model, the Listener and Server portions of the application coexist in the same CICS or IMS TP and run as part of the same CICS task. The server application, therefore, holds the socket until all application processing has completed. This means that after a client TP starts a server TP, another client TP cannot access the Listener or the server TP until the first client is finished.

- Concurrent model. In the Concurrent model, the Listener and Server portions of the application run under the control of different tasks. The Listeners purpose is to accept the connection and invoke the Server task. The Server portion of the application deals with all sending and receiving of application data and performing application-dependent processing. This model allows a higher degree of transaction concurrency because the listening socket is not held by a single client and can instead listen concurrently to multiple clients. Even though the CICS MS Link using TCP/IP programming model is not called concurrent, the server TP does work concurrently instead of iteratively.

In the four TI-supported TCP/IP models other than IMS Connect, there is both a TCP/IP connection Listener aspect and a Server aspect. The manner in which the Listener and Server portions of the application are implemented determines whether the Iterative or the Concurrent access model is used. The Concurrent access model requires the use of a Transaction Request Message (TRM); the Iterative model does not. The TRM is a formatted data record that identifies the IMS or CICS transaction program (TP) to be invoked and its characteristics.

In This Section

Iterative Model

Concurrent Model

See Also
Other Resources
Programming Models

Iterative Model

In the Iterative model, the Listener and Server portions of the application coexist in the same CICS or IMS TP, so the TP holds the socket until all application processing has been completed. The Iterative model uses this sequence:

1. Create a socket
2. Bind it to a local address
3. Listen (make TCP/IP aware that the socket is available)
4. Select (wait for a connection request)
5. Accept the connection request
6. Read or write the data
7. Close

The advantages of the Iterative model are:

- Simplicity
- Reduced network overhead and delay because a TRM exchange sequence is not required
- Less CPU intensive
- Higher single-threaded transaction throughput

The disadvantages of the Iterative model are:

- Severely limits concurrent access to TPs that run for a long time
- Server application contains all of the SEAPI calls (Create to Close)
- Each TP has its own Listener, which means duplicate code
- Select with timeout causes a CICS region to sleep

See Also

Other Resources

Iterative vs. Concurrent TCP/IP Models

Concurrent Model

In the Concurrent model, the Listener and Server portions of the TP run under the control of different tasks. The Listener's sole purpose is to accept the connection and spawn the Server task. The Server portion of the application sends and receives data and performs TP-dependent processing. This model allows a higher degree of concurrency because the listening socket is not held for an extended period of time.

The Listener must receive a TRM as the first data from the TI run-time environment. The TRM tells the Listener which TP to invoke and the characteristics of that program. After sending the TRM, the TI run-time environment must wait for a response before sending data. The Listener of the Concurrent model follows this sequence:

1. Create a Listening socket
2. Bind it to a local address
3. Listen (make TCP/IP aware that the socket is available)
4. Select (wait for a connection request)
5. Accept the connection
6. Read the TRM
7. Check the validity of the requested transaction ID (TRANID)
8. Give a socket (prepare TCP/IP for the transfer of the socket)
9. Start the task
10. Synchronize on the worker task acceptance of the socket
11. Select (wait for connection request)

The Worker task of the Concurrent model follows this procedure:

1. Take a socket (accept the socket request from the Listener).
2. Write a response to the TRM.
3. Read or write application data.
4. Close.

The advantages of the Concurrent model are:

- Easy to implement concurrent access to TPs that run for a long time.
- One Listener is shared by many TPs.
- Server TCP/IP logic is simple.

The disadvantages of the Concurrent model are:

- Increased network overhead and delays due to the requirement of the TRM exchange.

- More CPU and resource intensive than is the Iterative model.

See Also

Other Resources

Iterative vs. Concurrent TCP/IP Models

Host-Initiated Processing

Host-initiated processing (HIP) enables a host application to call a method of a COM or .NET object, pass parameters to the method, and receive parameters back from the method. As data travels first from the host to the client and then from the client to the host, the data is transformed from a format understandable by the host to the format appropriate for the client

Host-initiated processing is implemented in the following steps:

1. The HIP service process, called an application, begins listening for incoming connections on a list of endpoints specified by a local environment definition.

2. The client application, running on the host, initiates a TCP connection to a HIP system using one of the endpoints

3. The HIP service process checks if there is an established association between the endpoint and the clients host name or IP address. If no association is found, the connection is aborted.

4. The association uniquely identifies the work plan that is a sequence of workflows to be performed to complete the clients request. There are three types of work plans:

    a. Endpoint

    b. Transaction Request Message

    c. Data.

Endpoint

The endpoint work plan consists of a single final workflow. The association is directly mapped to a method of a COM object that is to be invoked for the clients request processing. The endpoint workflow performs the following:

1. Receives client data

2. Unpacks data and populates parameters for the method

3. Creates the object and calls the method

4. Packs returned parameters into the client data

5. Sends client data

6. Closes connection.

Transaction Request Message

The Transaction Request Message (TRM) work plan consists of two workflows: TRM and Final workflow. TRM workflow handles initial part of the conversation when client sends the TRM and the workflow replies with the TRM Reply. Depending on the type of TRM, the TRM workflow can use one of three TRM Handlers: Microsoft Concurrent Server, Microsoft Link, IBM Concurrent Server. The TRM workflow performs the following:

1. Receives and unpacks the TRM using the assigned input format

2. Passes the TRM to the assigned handler

3. The handler returns the resolution information and the positive TRM Reply

4. The resolution information, which is expected to be character data, is converted to Unicode using the code page associated with the host environment

5. Workflow queries the database if there is a mapping to a method of an object defined for the resolution information
6. In case the match is not found, the handler is called to get the negative TRM Reply
7. The TRM Reply is packed using the assigned output format and sent. In the negative case, the connection is aborted
8. Workflow passes control to Final workflow along with the method identity found

Data

The Data determinant work plan consists of two workflows: Data Determinant and Final workflow. Data Determinant workflow preprocesses the client data trying to find a match to one of the determinants defined for the association. Determinant includes a string of characters and the position of the string in the client data. Each determinant is mapped to a method of an object. At startup the determinants are pre-converted to all code pages of hosts associated with determinants. The rules are enforced that determinants for an end point – host association are not duplicated or one determinant is not a part of the other. The workflow follows these steps:

1. The list of determinants for the given End Point – Host combination is obtained and sorted by the sum of the length and position in ascending order
2. The first determinant is taken from the list
3. A part of client data is received
4. The data is checked if it matches the determinant
5. In case of no match the next determinant is taken from the list, more client data received if needed, data compared to determinant
6. When no available determinant matches the data the connection is aborted
7. If a determinant is found the control is passed to Final workflow along with the method identity and the client data already read. There could be a situation when the determinant data is located within or even after the client data mapped to the methods parameters.

Note

A HIP MVS client must perform a socket shutdown immediately after a socket send and before a socket receive. Failure to perform an immediate socket shutdown causes the TI runtime to refuse the data, break the connection, and log an 808 event message (**A Transaction Integrator HIP application received more data than expected.**) to the server application event log. In addition, the packet containing the data being sent to the workstation might appear misleading: the Microsoft Network Monitor shows the data within the packet and data length not being excessive.

See Also

Other Resources

Windows-Initiated Processing

Windows-Initiated Processing

Windows-initiated processing (WIP) enables Windows-based client applications to invoke mainframe based transaction programs (TPs). Transaction Integrator (TI) along with the Windows operating system includes everything you need to build, deploy, and manage TI components that enable mainframe transaction programs (TPs) to interoperate with Windows-based COM and .NET applications.

The following programming models support Windows-initiated processing:

- TCP Transaction Request Message Link
- TCP Enhanced Listener Message Link
- TCP Transaction Request Message User Data
- TCP Enhanced Listener Message User Data
- IMS Connect
- IMS Implicit
- IMS Explicit
- OS/400 Distributed Program Calls
- CICS LU6.2 User Data
- CICS LU6.2 Link
- IMS LU6.2 User Data

In This Section

What WIP Does

WIP Programming Model

Providing a Fail-Safe Environment for ACID Transactions

Using TI in a Non-DPL Environment

See Also

Other Resources

Transaction Integrator Architecture

What WIP Does

Windows-initiated processing (WIP) enables Windows-based client applications to invoke mainframe based transaction programs (TPs). Transaction Integrator (TI) provides a COM or .NET object interface (an Automation interface), maps data types to convert from the Intel-based architecture to the OS/390-based or OS/400-based architecture, and interacts with the host TPs.

In This Section

How TI Associates a Method with a TP

How TI Enables TPs to Return Exceptions

See Also

Other Resources

Windows-Initiated Processing

How TI Associates a Method with a TP

TI generally establishes some type of association between a method call in the Windows environment and the corresponding TP on the mainframe.

In the CICS TRM and ELM Link programming models, you can specify three names:

- Mainframe transaction program (TP) name
- TI component's method name
- Source TP Name

The name of the mainframe TP is constant for a given remote environment (RE). CSMI (the Mirror transaction) is an example of a mainframe TP Name. Each method name can be associated with a different mainframe TP Name. In addition, a third name, called the Source TP Name, can be specified for each method. The Source TP Name can be used with DB2 to associate a specific mainframe TP (and TI method) with a region control task (RCT).

In the CICS TRM and ELM User Data programming models, each method in a TI component is associated with a TP Name, and multiple methods can be associated with the same TP Name. If multiple methods are associated with the same TP Name, TI assumes that the TP is able to distinguish between the requests generated by each method. The TI run-time environment should optionally provide Meta data to assist the TP with this process. When Meta data is included, the method name is sent as a fixed-length character string (32 characters) and is always the first item sent. The method name is left justified in the field and padded with blanks. Developers can specify the method-TP Name mapping on a per-method basis or as a default for the entire component.

When COBOL that is imported to build a TI component library contains REDEFINES clauses, the TP can expect different kinds of requests. The developer should consider creating a method for each REDEFINES group that represents a different message format. All of the methods created from these REDEFINES groups can map to the same TP.

See Also
Other Resources
What WIP Does

How TI Enables TPs to Return Exceptions

TI provides a Meta data mechanism for returning exceptions from Automation server applications like TI applications. TI uses this mechanism to provide the mainframe developer with an optional way to return mainframe error information (also known as exception data) back through the normal application.

A transaction program (TP) returns error information as optional Meta data that includes an exception block as part of the reply message. The exception block contains information, in a standard format, that can be used to populate an Automation exception structure.

TI error messages have numbers in the range 0-9999. Meta data error message numbers returned from the mainframe can fall within the same range. To distinguish TI error messages from Meta data messages returned from the mainframe, TI adds 10000 to the number of any Meta data error message returned from the mainframe.

A TP can also use this mechanism to provide information regarding the TP state to the TI run-time environment. Specifically, a TP can indicate whether the TP:

- Is willing to commit the work performed so far (and deallocate the conversation).
- Can perform no more work on the current conversation and expects the client to prepare and commit.
- Has encountered an error that will prevent it from committing the transaction.

While it is always possible for a TP to deallocate the conversation abruptly, TI exceptions allow it to return detailed information about the error to the calling client application.

TI uses the information contained in the exception block to update state information in the TI run-time environment and (if requested) return an exception to the client application.

The following table shows the fields in the EXCEPINFO exception structure.

Field	**Description**
wCode	The error code returned in the exception block.
bstrSource	Generated automatically by TI based on information about the customer's object and the remote TP.
bstrDescription	From the exception block. This error description comes from the remote TP.
bstrHelpFile	Formed by taking the Help path associated with the object's component library (in the Registry) and combining it with an unqualified file name included as custom information in the component library. This allows the developer to identify the file name of the Help file as it was created, while giving the administrator ultimate control over where the Help file is installed during deployment.
dwHelpContext	From the exception.
scode	Same as *wCode*.

It is possible for the TP to return state information without actually raising an exception. To keep the mainframe TP code as straightforward as possible, the exception data is part of the optional Meta data and is returned in all cases, whether an error occurs or not.

See Also

Other Resources

What WIP Does

WIP Programming Model

The programming models provide a synchronous bridge between the component object model (COM) or the .NET Framework and the mainframe transaction-programming model. Consequently, Transaction Integrator (TI) has no APIs that a developer must use.

Although TI uses existing mainframe programming models, you may need to make some changes to an existing mainframe transaction program (TP) if any of the following are true:

- The TP uses a conversational or pseudo-conversational mode. TI supports only the nonconversational TP model known as the ping-pong or request-reply conversational sequence in conversations between clients and servers. The TI programming model requires nonconversational method calls; that is, a single input message and a single output message. See "Supported Conversational Model" for more information.

- The TP has terminal processing logic embedded in the same program with the business logic. To get this program to work with TI, you must first restructure it as two separate TPs, one for the terminal processing logic and the other for the business logic. Then you can use TI with the business logic TP.

- A CICS Link transaction program (TP) using LU 6.2 uses explicit EXEC SYNCPOINT commands. There may be a way to work around this issue without rewriting the TP. For more information, see TPs with Explicit SYNCPOINT Commands.

The topics in this section give you the details on the mainframe programming models and how they are addressed in the TI programming model.

In This Section

Supported Conversational Model

TPs with Explicit SYNCPOINT Commands

Support for Transactions and Two-Phase Commit

See Also

Other Resources

Windows-Initiated Processing

Supported Conversational Model

TI supports only the nonconversational (ping-pong) model. A mainframe transaction program (TP) may not expect a request-reply (ping-pong) nonconversational sequence like that required by TI, in which case, you will need to modify the mainframe TP. The mainframe TP may be set up to communicate with other TPs by using the conversational or pseudo conversational models, neither of which are supported by TI.

In a nonconversational transaction, the entire exchange of data between the client and server occurs within a single method call. The exchange of data can be fairly complex (for example, if recordsets are used), but the base application has no opportunity to delay the exchange of data or otherwise cause the conversation to remain idle.

A pseudo-conversational TP attempts to get conversational-type client/server interactions while avoiding the scalability problems associated with true conversations. Pseudo-conversations are implemented by TPs that maintain state for clients over a series of nonconversational requests. An application-specific context handle is used to retrieve the saved state over the course of the pseudo-conversation. TI does not support the pseudo-conversational model.

See Also

Other Resources

WIP Programming Model

TPs with Explicit SYNCPOINT Commands

CICS LU 6.2 Link transaction programs (TPs) cannot use explicit EXEC SYNCPOINT commands to control the transaction semantics of COM+ transactions. However, if an existing CICS Link TP does issue explicit EXEC SYNCPOINT commands, you do not need to modify the TP to have it work successfully with TI. You need only meet the following two requirements for TI components that execute within the transaction.

- Set the TI component's transaction property to **Does not support transactions**. You define a component's transaction property when you create the component in TI Designer. Once you deploy the TI component in a COM+ application, you can view or modify its transaction property in TI Manager.

- Configure the remote environment (RE) that describes the region on the mainframe and that hosts the CICS Link TP to allow the use of explicit SYNCPOINT commands for non-transactional components.

To configure the RE to allow the use of explicit SYNCPOINT commands

1. In TI Manager, right-click the RE that you want to configure, click **Properties**, and then click the **CICS Mirror TP** tab.
2. Select the **Allow use of explicit SYNCPOINT commands for nontransactional components** check box.
3. Click OK.

If the two requirements are not met, the transaction will not work on either the Windows or the mainframe side. In which case, TI writes a message to the Windows 2000 Event Log explaining the cause of the failure.

See Also
Other Resources
WIP Programming Model

Support for Transactions and Two-Phase Commit

In COM terminology, a transaction is always a unit of work that is atomic, consistent, isolated, and durable (ACID). In mainframe terminology, a transaction may or may not be an ACID transaction; in mainframe terminology, a transaction is a set of operations or commands in a transaction program (TP). This difference in terminology can be confusing. The word transaction as it is used in TI Manager and TI Designer always refers to an ACID transaction.

Two-phase commit (2PC) is a protocol that allows a set of application (or cross-application) operations or commands to be either all rolled back or all committed as a single transactional unit.

Note
If you invoke a TI Automation server over the TCP/IP protocol, there is no support for two-phase commit transactions. Two-phase commit works only over the SNA APPC/LU 6.2 protocol.

A TI component has four possible transactional properties:

- Requires transaction
- Requires new transaction
- Supports transactions
- Does not support transactions

The first two choices require the mainframe TP to be transactional (that is, meet the ACID properties) and to support Sync Level 2. This is transparent to the mainframe TP if it is a CICS Link or IMS version 6.0 or later program. The third choice requires the mainframe TP to support Sync Level 2 requests and handle the transaction semantics appropriately. The fourth choice is required for IMS TPs prior to IMS version 6.0 and for any CICS TPs that support only Sync Level 0 or Sync Level 1.

If a TI component is invoked within the scope of a COM+ transaction, TI will establish a Sync Level 2 conversation with CICS (otherwise, Sync Level 0 is used). This is transparent to the client of the TI component. If the mainframe TP is a CICS Link program, the transactional nature of the conversation is transparent to the TP as well, since IBM's mirror transaction in CICS (CSMI) handles the Sync Level 2 protocol, and the TP to which it is linked is unaware whether Sync Level 0 or Sync Level 2 is being used.

TI complies with the COM+ programming model by calling SetComplete or SetAbort when it completes the operation of each method call from the client. If no errors were detected, TI calls SetComplete; otherwise it calls SetAbort. TI also calls SetAbort if the mainframe TP indicates that the transaction should not commit by setting the DisableCommit flag in the meta data error block returned. TI Automation client applications can also choose to call SetAbort if they determine that there are application-level problems that should prohibit the transaction from committing.

When the client's method call returns, the TP on the mainframe has performed some unit of work, but any changes to protected resources in CICS are not yet committed. TI uses new DTC interfaces to enlist the Sync Level 2 conversation on the DTC transaction. When DTC is ready to commit or abort the transaction, it communicates with TI to drive the appropriate two-phase commit flows on the LU 6.2 conversation. Again, all of the 2PC work is performed transparently by TI on the client's behalf.

Although the TI object can be deactivated when the method completes, the conversation must be maintained until the transaction commits or aborts. Users can adversely affect performance and tie up system resources if their application code makes one or more transactional method calls but does not commit the transaction for a long period of time. Conversations can be quickly consumed by poorly structured user code.

When a conversation is waiting to commit, it will be divorced from the object with which it was associated. TI manages a pool of these "waiting" conversations and performs the required sync-level operations when the appropriate notifications are received from DTC. When possible, TI reuses these conversations to minimize overhead.

TI also provides a resynchronization service (SNA LU 6.2 Resync TP). This Windows 2000 service is configured to be the auto-started invokable service for the SNA-defined Resync TP (0x06f2). The Resync service implements the "Exchange Log Names" and "Compare States" functions of an SNA transaction manager. It allows both DTC (Distributed Transaction Coordinator) and CICS to initiate the recovery process as required during system startup or following a system or communication failure.

For information about IBM's SNA SyncPoint or 2PC flows, *SNA SyncPoint Services Architecture Reference (IBM SC31-8134-00)*. All TI 2PC flows are implemented in conformance with this architecture.

Note
For information on how to use CICS Link TPs that use explicit SYNCPOINT commands, see TPs with Explicit SYNCPOINT Commands.

In summary, to use two-phase commit, you must meet all of the following requirements:

- The local and remote LUs must each have SyncPoint support enabled in the Host Integration Server node.
- The local and remote LUs should each point to the computer that is running Resync services.
- The remote environment (RE) must have Sync Level 2 support enabled. To check this, right-click the RE in TI Manager, click Properties, and then click the LU 6.2 tab.
- The TI component must have Transaction Support set to Supported, Required, or Requires New. To check this setting, right-click the TI component in TI Manager, click Properties, and then click the Transactions tab.
- The remote host computer must be configured for Sync Level 2 support.

See Also
Other Resources
WIP Programming Model

Providing a Fail-Safe Environment for ACID Transactions

ACID (atomic, consistent, isolated, and durable) transaction processing using two-phase commit (2PC) generally requires a fail-safe environment, which is an environment that ensures continuation in spite of hardware failures. This is often called 2PC failover or hot backup.

Host Integration Server includes enhancements to the **SNA LU 6.2 Resync TP** commonly called the Resync service along with enhancements to the configuration and APPC DLL to make 2PC failover work through two or more redundantly configured Host Integration Server SNA servers (computers). In the event of a failure of one of the servers (computers), a separate Host Integration Server computer running either TI or the DB2 Provider can continue to initiate transactions through an alternate server (computer).

To configure 2PC failover to work with Host Integration Server, complete these tasks:

- Configure two Host Integration Server servers to support the same SyncPoint-enabled local APPC LU alias but with different LU names. Have these local APPC LUs point to the same computer name where Microsoft Distributed Transaction Coordinator (DTC) service and the Resync service are running (that is, a separate Host Integration Server computer that supports TI or an application that uses the DB2 Provider). Also, have both servers support the same remote APPC LU alias and name.

- In the applicable TI remote environment (RE), configure the local and remote LU aliases, and choose transactional support. If the application is using the DB2 Provider, configure the Universal Data Link with the local and remote APPC LU aliases, and set the **Units of Work** property to DUW.

When the Resync service starts, it searches all SyncPoint-enabled local APPC LUs that specify the computername where the Resync service is running. Resync then initiates an Exchange Log Names request over every found local APPC LU with all SyncPoint-enabled remote APPC LUs.

When a TI Automation server (application) or the DB2 Provider invokes a transaction program (TP) on the mainframe and initiates a conversation, the APPC DLL locates any available Host Integration Server server (computer) that supports the LU/LU pair.

In this way, a TI Automation server (application) or the DB2 Provider gains fault tolerance by getting a conversation through any Host Integration Server server (computer) that supports the LU/LU pair. The Resync service then coordinates the DTC transaction log reconciliation when a Host Integration Server SNA server (computer) comes back online, if a server (computer) failure occurs during a transaction. Note that this configuration does not provide fault tolerance for the Host Integration Server server (computer) that is running only TI or the DB2 Provider, not the SNA service.

Note
Clustering the servers (computers) that are running the SNA service is not recommended. Instead of using Windows Clustering, use the configuration recommendations described in this topic.

Note
2PC works only when you are using the SNA (APPC/LU 6.2) protocol to communicate with the host system. Neither TI nor the DB2 Provider support 2PC over the TCP/IP transport, so there is no 2PC failover solution for TCP/IP-based systems.

See Also

Other Resources

Windows-Initiated Processing

Using TI in a Non-DPL Environment

A non-linked environment (that is, a non-DPL environment) is one that does not use IBM Distributed Program Link (DPL). You can use TI to invoke a mainframe transaction program (TP) that uses the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands. These two COBOL commands are useful when you want a CICS TP to take on SNA (APPC/LU 6.2) conversation responsibilities and therefore bypass the Mirror TP. In other words, the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands are most often used in a non-linked environment to transfer data to and from an LU of type 6.2 (APPC).

TI supports the LU 6.2 model for both linked and nonlinked environments. You can create the following remote environment (RE) types to support each model:

- **CICS Link using LU 6.2**

  Use this in an IBM DPL environment that uses the Mirror TP.

- **CICS using LU 6.2**

  Use this in a non-DPL environment that bypasses the Mirror TP.

Many customers use TI in a non-DPL environment. The only concern is whether terminal logic is embedded with the business logic. When a COBOL TP supports IBM DPL, the presentation logic has already been separated from the business logic, so you probably will not need to modify the COBOL. However, if the TP was written to communicate with a terminal, it is likely that you will need to modify the COBOL code to separate the business logic from the presentation logic.

For example, the following sample COBOL code shows how to handle unbound recordsets by using the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands:

```
*****************************************************
* Example showing how to send unbounded recordsets
* to a client application.
*****************************************************
 IDENTIFICATION DIVISION.

 ENVIRONMENT DIVISION.

 DATA DIVISION.

 WORKING-STORAGE SECTION.

* INPUT AREA
 01  CUSTOMER-INPUT-NUMBER               PIC 9(9).

* OUTPUT AREA
 01  CUSTOMER-DATA.
     05  LAST-NAME                       PIC X(20).
     05  FIRST-NAME                      PIC X(20).

* ONE ROW IN A DATABASE TABLE
 01  INVOICES.
     05  INVOICE-NUMBER                  PIC 9(10).
     05  INVOICE-DATE                    PIC 9(7) COMP-3.
     05  INVOICE-AMOUNT                  PIC S9(13)V9(2) COMP-3.
     05  INVOICE-DESCRIPTION             PIC X(40).

 LINKAGE SECTION.

 PROCEDURE DIVISION.
*
*   Get the input customer account number from the
*   client applicaton:
*
     MOVE LENGTH OF CUSTOMER-INPUT-NUMBER TO RECEIVE-LENGTH
```

```
      EXEC-CICS RECEIVE INTO(CUSTOMER-INPUT-NUMBER)
                LENGTH(RECEIVE-LENGTH)
                END-EXEC.
*
*    Do some work; then send the first and last name back:
*
      MOVE LENGTH OF CUSTOMER-DATA TO SEND-LENGTH
      EXEC-CICS SEND FROM(CUSTOMER-DATA)
                LENGTH(SEND-LENGTH)
                END-EXEC.
*
*    Follow regular data with unbounded table data that
*    the client application sees as a recordset:
*
      MOVE LENGTH OF INVOICES TO SEND-LENGTH
      PERFORM UNTIL        NO-MORE-INVOICES
*
*    Do some work to get the next row:
*
      MOVE DB-INVOICE-NUMBER TO INVOICE-NUMBER
      MOVE DB-INVOICE-DATE   TO INVOICE-DATE
      MOVE DB-INVOICE-AMOUNT TO INVOICE-AMOUNT
      MOVE DB-INVOICE-DESC   TO INVOICE-DESCRIPTION
*
*    Send the row:
*
      EXEC-CICS SEND FROM(INVOICES)
                LENGTH(SEND-LENGTH)
                END-EXEC.
      END-PERFORM.
```

See Also

Other Resources

Windows-Initiated Processing

Security and Protection

The information contained in the following sections details securing your Host Integration Server environment, including Enterprise Single Sign-On.

For information about Single Sign-On, see Understanding Enterprise Single Sign-On.

In This Section

Understanding Windows Security

Maximizing Product Security

Resource Security

Transaction Integrator Security

Understanding Windows Security

This section presents general security information.

In This Section

Security Overview

As corporate networks become increasingly integrated with the Internet and other external networks, the threat of unauthorized access to your corporate resources grows significantly. When you plan your networking environment, you must strike a balance between protecting your resources and allowing users unobstructed access to data.

The following list includes some of the questions that must be answered as you prepare to secure your Host Integration Server environment:

- How can I authenticate users who require host resources?
- How can I control access to host resources?
- How can I control sensitive data flowing between workstations and host applications?
- How can I maintain security while connecting my host network to external sources like the Internet?
- How can I make host resources easier to access for authenticated users, yet maintain a more secure and easy-to-administer host environment?

This section helps answer these questions and describes how to implement the security features to create a more secure LAN-to-host environment.

See Also

Other Resources

Understanding Windows Security

File System Choices and Security

One of the keys to controlling security on Host Integration Server computers is the configuration file. To maintain good security with this file, be sure to install the Host Integration Server server software on an NTFS (NT File System) partition. With NTFS, you can assign permissions on a file-by-file basis. With the FAT (File Allocation Table) file system or HPFS (High Performance File System), this is not possible. Installing Host Integration Server software on an NTFS partition is especially important for the primary server because this computer contains the master copy of the configuration file used by all Host Integration Server computers in the subdomain.

If you must install Host Integration Server on a non-NTFS partition, you can limit a users ability to view or change a Host Integration Server configuration file through the Host Integration Server SNA Manager, but you will not be able to set file permissions on the configuration file itself.

In addition, if you choose to install the software for one or more servers on a FAT or HPFS partition, you can set appropriate Read and Change permissions on the share, which gives servers access to the configuration file. (This is unnecessary for Host Integration Server software installed on an NTFS partition, because file permissions established through the Host Integration Server SNA Manager place tighter control on the configuration file than can be established by controlling the share.)

Securing Host Integration Server 2009 Files and Directories

If you install Host Integration Server onto an NTFS volume, you can control which SNA subdomain users can modify the Host Integration Server configuration file. A Host Integration Server administrator can control access to the configuration using the Host Integration Server SNA Manager. By specifying access permissions for users, you can control who has the ability to administer, change, or view a configuration.

When you install Host Integration Server, you create a single directory tree that contains the files needed to configure and use Host Integration Server. To control access to the configuration files in these shares, use the following guidelines:

- Create domain user accounts to run the SnaBase and SnaServer services. A domain user account is one that is allowed to log on to the network.
- Disable the Windows domain guest account.

At least one domain user or group must have Full Control permissions over all the shares, preferably a trusted group such as Administrators. If no user or group has Full Control permissions, the only person who can change the share permissions is the owner of the share. If necessary, this individual can change his or her permissions to Full Control as needed.

See Also

Other Resources

Understanding Windows Security

Authentication

When Host Integration Server receives a request to access a host resource, such as an LU for a terminal session, the server must have some way to verify the request.

User validation is a fundamental security issue that you can address using one of two methods: Domain Authentication and Workstation Authentication. The method you choose depends upon the type of service being requested.

Domain Authentication

A *domain*, as it pertains to Windows, is a group of computers that share a network resource database and have a common security policy. A Windows domain contains a *primary domain controller* (PDC) computer that acts as the resource and user manager for the entire domain. One or more computers in the domain can be configured to act as a *backup domain controller* (BDC). The BDC can take over for the PDC should any problems arise. The remaining computers in the domain are user workstations or servers that provide resources to domain users.

Within a Windows domain, Host Integration Server computers are logically grouped into an entity called a *subdomain*. Each SNA subdomain can contain up to 15 Host Integration Server computers, and a Windows domain can contain an unlimited number of subdomains. It is common to have multiple domains that manage user accounts independently of one another.

The domain model provides two key advantages over peer-to-peer networks with regards to security:

- You can manage user accounts from a central location.
- You can set up one unified security system for all user accounts in the domain.

Host Integration Server relies on the PDC or BDC to provide authentication services to users requesting access to host resources. Only users who have been validated by the PDC or BDC can gain access to resources provided by servers in the subdomain.

You can use domain authentication to verify users who request resources provided by these services:

- 3270 or 5250 terminal access from workstations using Host Integration Server client software
- APPC, CPI-C, or LUA applications built using Host Integration Server APIs

See Also
Concepts
Authentication
Workstation Authentication

Workstation Authentication

When you receive a request from a source that is external to the corporate network containing the SNA subdomain, or if you are not using a domain-type network, domain authentication is usually difficult to implement.

In this case, Host Integration Server performs workstation authentication on the following services:

- TN3270 terminal access
- TN5250 terminal access
- 3270 access from terminal emulators connecting through Host Integration Server client software

For these services, you can specify a list of allowed client workstation IP addresses for defined resources. When Host Integration Server receives a session request, it determines whether the requesting IP (or workstation name for TN3270E connections) matches that specified for the requested resource. Once verified, Host Integration Server allows the request to proceed.

Note
This type of authentication is not as secure as domain security because workstation names and the IP address are transmitted in clear text over the network.

See Also

Concepts

Authentication

Domain Authentication

Client Logon

The logon process conducted by a user on a client system performs the essential task of identifying the user to the Windows domain and to Host Integration Server computers in the domain. The way that Host Integration Server handles the logon process depends on the network software on the client system.

For any type of client computer, once the clients logon process for one Host Integration Server computer has been completed, additional servers can provide services without another interactive user logon. Servers can repeat the logon verification without the user being involved.

If you maintain tight security with the logon to a mainframe, it may make sense to keep access open-ended on the Host Integration Server computer.

See Also

Other Resources

Understanding Windows Security

Denying Access to Host Integration Server

You can use one of two methods to deny a user or group the ability to use Host Integration Server. Either remove the user or group from the list of those who have permissions, or assign No Access to the user or group.

When removing a user or group from the list of those who have been granted permissions, be sure to consider all groups to which a particular user belongs. To deny access to that user, either remove the applicable groups from the list of those with permissions, or remove the user account from these groups. Otherwise, the user can receive access through one of the groups (unless the second method, assigning No Access, is used).

You can deny all access to a user or group by assigning No Access, which overrides any other permissions that may apply to a user. For example, if a user belongs to a group that has Read permission, but the user has No Access, the user is not permitted administrative access. Likewise, if a user belongs to Group1 and Group2, and Group1 has Read permission while Group2 has No Access, the user is not permitted administrative access.

The following table provides recommendations for using the two methods of denying access.

Method of denying access	**Recommended situations for use**
Remove from list of those granted permissions	Use when you want to deny access to a medium or large group. If you then want to grant access to specific users in that group, add those users to the list of those with permissions. In contrast, if you assign No Access to a large group, none of the members of the group will be allowed access, even if explicit access is assigned to them through other accounts.
Assign No Access	Use when you want to deny access to an individual user or a small group. If you then want to grant access to other groups to which the affected users belong, you can do so. The No Access permission overrides the other permissions for the affected users. In contrast, if you attempt to deny an individual user access by removing the user from the list of those granted permissions, a group to which the user belongs might be left in the list. The permissions assigned to that group would be given to the user.

See Also

Other Resources

Understanding Windows Security

Security Audit

When you set up security auditing, you specify events for which the system will create event log entries.

You can monitor system events using the Host Integration Server SNA Manager. When you configure the properties, you can view a detailed problem analysis, significant system events or general information messages, or you can disable audit logging.

You can cause the system to create an entry every time a member of the Administrators group changes the configuration file. The entry would include the time, the user and process IDs, and other information.

When you set up auditing with Host Integration Server, you are able to record not only access to the configuration file, but the running of Host Integration Server processes. However, using a file system other than NTFS limits the possibilities for auditing, just as it limits the possibilities for setting permissions. When Host Integration Server is not installed on NTFS, the only event that will actually result in log entries is the starting of the Host Integration Server SNA Manager. However, auditing information for this event is useful, if limited.

Auditing is an important way of collecting security-related information. However, there is a small performance overhead for each audit check the system performs. In addition, if you record a wide variety of events, the event log can grow very quickly. For these reasons, you may want to be selective when specifying events to be audited. You may want to focus on such events as Read Domain (Success), Write Domain (Success), and Manage Domain (Success). This list of events is a suggestion only. All the events offered for auditing through Host Integration Server can be useful, depending on the situation.

Among the events recorded when you enable auditing are the actions of Host Integration Server itself. For more information about event logs, the Event Viewer, and strategies for auditing, see your Windows documentation.

See Also

Other Resources

Understanding Windows Security

Viewing and Interpreting Audited Events

Events audited by Host Integration Server are recorded on the server in which they occur. They are not stored centrally, in contrast with other records generated through Host Integration Server.

The records generated through auditing are stored in the security section of the event log. In contrast, other Host Integration Server events are recorded in the application section of the log.

Note
When you view log entries for audited events related to Host Integration Server, you will see a record of actions carried out by Host Integration Server itself, as well as actions carried out by users. The source of the actions carried out by Host Integration Server is the account under which the service is running.

The event details for some events will list an item called **Accesses** in the **Description** box. Unusual information may be shown here for events related to Host Integration Server.

Use the following table for interpreting this information.

Information shown for Accesses (under Description)	**Meaning**
Unknown specific access (bit 0)	Read Domain access. This access applies when someone starts or attempts to start the Host Integration Server SNA Manager. (Host Integration Server must read the configuration file in order to function.)
Unknown specific access (bit 1)	Write Domain access. This access applies when someone starts the Host Integration Server SNA Manager in write mode — that is, when someone starts Host Integration Server and the primary Host Integration Server computer is available.
Unknown specific access (bit 2)	Manage Domain access. This access applies when someone starts or attempts to start the Host Integration Server SNA Manager.

See Also

Other Resources

Understanding Windows Security

Firewall Support Overview

Host Integration Server client computers and servers can interoperate with screening routers, where the client computer knows the address of the Host Integration Server computer. Host Integration Server client computers and servers can also interoperate with full-blown Internet firewalls, where the end user is not allowed to know the IP address of the Host Integration Server computers.

A firewall is a network security device that restricts access to network resources by allowing traffic only through specified port numbers. These port numbers are commonly defined in software components to allow Internet firewalls to filter packets based on port number, thereby denying or accepting their propagation to the private network. In many instances, the services of a firewall are provided in conjunction with a network router that bridges two network segments together.

Host Integration Server can be configured to use specific software port numbers, allowing administrators of Internet firewalls to filter packets based on port number. By assigning specific destination port numbers to Host Integration Server components, Host Integration Server can interoperate with screening routers and full-blown firewalls.

If the Host Integration Server address is known, the client workstation configures the appropriate port and destination IP of the Host Integration Server computer in the client software, for example, 1477 and 128.124.1.2. Alternatively, the Host Integration Server computer's service port numbers can be changed to the port number requested by the client.

You can configure destination port numbers in Host Integration Server for end-user and Administrator clients using TCP/IP, IPX/SPX.

Each network transport has the following three components for which you can assign destination ports.

DatagramPort

This port is used for datagram (mostly broadcast or multicast) traffic. Use the **Server Broadcasts** dialog box in the Host Integration Server SNA Manager to control which transport is used for server-to-server communication.

SnaBasePort

This is the port to which the server SnaBase listens for new client sponsor connections. Sponsor connections are used by the Host Integration Server client to learn about the SNA subdomain in which it participates.

SnaServerPort

This is the port where the SNASERVR.EXE waits for new application session requests.

If the Host Integration Server address is not known, the Host Integration Server IP transport replaces the real destination IP address with the address of a firewall. The firewall then maps the connection request to the actual Host Integration Server computer. This takes place when the transport opens a connection to a Host Integration Server-based computer for application sessions or a sponsor connection.

Host Integration Server supports firewalls primarily on TCP/IP networks. It may also be possible to implement firewalls on IPX/SPX. For more information on configuring a firewall, consult your network documentation.

Note
The clients obtain the Host Integration Server ports through the sponsor connection. On IPX/SPX, the clients get the SnaBase ports from the NetWare bindery and VINES StreetTalk, respectively. There is no need to configure any port numbers on these two client network types.

It is also possible to create IP mappings which, by overriding service table entries, replace a specified IP address with a firewall address. To create a mapping, add an IPMapping key under the SnaTcp key, and then add a REG_SZ value for each mapping. An example is shown below:

```
IPMapping
xxx.xxx.x.xxx = yyy.yy.yy.yyy
```

With this mapping, any attempts at establishing a connection with *xxx.xxx.x.xxx* will instead connect to *yyy.yy.yy.yyy*.

For additional information on firewall support, including client-side registry settings and the use of DLS with firewalls, see the following Knowledge Base articles: Q139508, Q164590, Q224303, and Q215838.

See Also

Other Resources

Understanding Windows Security

Screening Routers and Internet Firewalls

You can configure destination port numbers in Host Integration Server for End-user and Administrator clients using TCP/IP, IPX/SPX.

Each network transport has the following three components for which you can assign destination ports.

DatagramPort

This port is used for datagram (mostly broadcast or multicast) traffic. Use the **Server Broadcasts** dialog box in Host Integration Server SNA Manager to control which transport is used for Server-to-Server communication.

SnaBasePort

This is the port to which the server SnaBase listens for new client sponsor connections. Sponsor connections are used by the Host Integration Server client to learn about the SNA subdomain in which it participates.

SnaServerPort

This is the port where the SNASERVR.EXE waits for new application session requests.

See Also

Other Resources

Understanding Windows Security

Data Encryption

Data Encryption helps secure traffic between the client computer and Host Integration Server on a per-user basis. There is a client component and a server component. The data encryption is implemented transparently to any application that is written to the Host Integration Server APIs (application program interfaces). Any software, such as a third-party emulator, that is written to use the Host Integration Server client APIs will automatically benefit from the encryption.

Host Integration Server lets you encrypt data for client-to-server and server-to-server communications.

Client-to-server encryption prevents information from being sent in clear text between Host Integration Server client workstations and Host Integration Server computers. Data encryption enhances network security on the client-to-server communications path for all applications using Host Integration Server client connections, including 3270/5250 emulators and APPC logon IDs and passwords. Data encryption is enabled by default.

Server-to-server encryption can be used to help provide more secure communications across your network, the Internet, or any other wide area network. If a user enables data encryption, information transferred through Distributed Link Service (DLS) is automatically more secure.

Data encryption is also supported in the distributed deployment of SNA Open Gateway Architecture (SOGA), which uses Host Integration Server to help provide more secure server-to-server communications across the Internet, intranets, and other WANs.

Host Integration Server lets you encrypt data for client-to-server and server-to-server communications.

Client-to-server encryption prevents information from being sent in clear text between Host Integration Server client workstations and Host Integration Server computers. Data encryption enhances network security on the client-to-server communications path for all applications using Host Integration Server client connections, including 3270/5250 emulators and APPC logon IDs and passwords. You can enable data encryption on a user-by-user basis using the Host Integration Server SNA Manager.

Server-to-server encryption can be used to provide more secure communications across your network, the Internet, or any other wide area network. If a user enables data encryption, information transferred through the Distributed Link Service (DLS) is automatically secure.

Data encryption is enabled for Distributed Link Service by adding the domain user account under which Host Integration Server services such as SnaBase or SnaServer are running to the SNA subdomain. The actual encryption is implemented in the transport providers layer of the Host Integration Server architecture. You can then enable data encryption settings for the user account, as described above.

See Also

Other Resources

Understanding Windows Security

Maximizing Product Security

In addition to the general guidelines elsewhere in this section, the following specific recommendations can help you increase the security of your Host Integration Server deployment. Since all of these actions are performed during deployment or configuration, procedures are located in the appropriate sections of this documentation.

Note
Whereas these recommendations apply across the entire product, the Transaction Integrator Threat Mitigation section also of fers information specifically for TI users.

Single Sign-On

The most effective action you can take is to use integrated security by using the Host Integration Server Single Sign-on (SSO) feature. It is especially important to use SSO because certain data used by the product is impossible or unwise to encrypt, making it potentially vulnerable. The SSO feature reduces or eliminates this issue.

Accessing SQL Server Databases

When you are accessing a SQL Server database:

- Because the Resync service will access this database -- and the database may service multiple, unsecured domains -- you should always use a local SQL Server database and never grant remote access to it.
- Use only Windows NT integrated security, and restrict access to only privileged Windows NT accounts.
- Use only Host Integration Server security groups which were created with the Host Integration Server Configuration Wizard.

When you are connecting via SNA Protocol:

- When connecting to an upstream Host Integration Server computer, use client/server encryption.
- When implementing SNA Open Gateway Architecture (SOGA) Distributed Link Service (DLS) across wide area networks, use server-to-server encryption.
- Locate upstream Host Integration Server computers within the data center using secure Token Ring, Ethernet, Bus and Tag Channel, or ESCON fiber channel attachments.

When you are connecting via TCP/IP protocol:

- Use upstream Windows software router computers or hardware routers to encrypt the TCP/IP traffic.
- Locate the upstream router within the data center using either secure Token Ring or Ethernet connections to the host.
- Because DRDA AR sometimes stores data in plain text:
    - Use direct network connections over a private segment, with static IP or SNA addresses.
    - Use IPSec where supported for IP communications between host and Windows servers.
- When connecting an SNA LU6.2 network to a mainframe or AS/400, with the Host Integration Server computer deployed as the SNA gateway to a downstream Host Integration Server computer, use Host Integration Server server-to-server data encryption.
- For SNA LU6.2 network connections to mainframe, use the IP-DLC Link Service in conjunction with IPSec.
- Use encryption that is part of the Host Integration Server server-to-server and client-to-server connections.

- When connecting to a mainframe DB2 for z/OS, use IPSec on an IP-DLC, and also use NNS on the target system to utilize direct connections to DLUS and APPN Peer resources.

When you are accessing DB2 DUW over the IP Resync Service:

- Increase the level of security access rights to the SQL Server database which stores the UOWID mapping tables.
- Use the (default) separate SQL Server database, as opposed to one shared with other applications.

To protect unencrypted data and credentials in the com.cfg file:

- Implement IPSec.
- Deploy the Host Integration Server computer in an isolated network segment.
- Increase security settings on the host account used for session security.

When using the TN3270 Server:

- Stop and restart the TN3270 Server whenever a new CRL is downloaded. Otherwise, you will be using an out of date CRL, which could permit unwanted access to the host.

Server-to-Host Security

The following actions will increase server-to-host security, especially on an APPN Network or UDP sockets for HPR/IP Protocol Traffic:

- Deploy Host Integration Server in a secure network segment, and use encryption that is part of the Host Integration Server server-to-server and client-to-server connections.
- Use IPSec on the IP-DLC connection.
- Use NNS on the target system to utilize direct connections to DLUS and APPN Peer resources.
- Use a direct IP-DLC Connection to CS/390 (DLUS) and NNS, or a direct IP-DLC Connection to a peer APPN node.

Additional Security Recommendations

Finally, as in the following recommendations, be vigilant about access to every file, connection, or other product component:

- In general, use System or User DSNs rather than file DSNs, as System and User DSNs are more secure.
- When using the OLE DB Provider for AS/400 and VSAM, store your HCD files in a secure local folder or share that can only be accessed by the developer and the runtime application.
- When using Transaction Integrator, place any objects going to CICS or IMS in a remote environment that requires Enterprise Single Sign-On.
- Be vigilant about your access control list (ACL). Although it is possible to install Host Integration Server and inherit a previous ACL, you should remove any existing ACLs and replace them with new ones.
- Store Printer Definition Tables (PDTs) and Printer Definition Files (PDFs) in a secure location to prevent their being replaced with a rogue file.
- Since trace files may contain non-encrypted data, always store them in a secure location, and delete them as soon as trace analysis is complete.

- Minimize unwanted access to the Resync Service by running it on the same computer as the application it services.

- Enable LUA Security for TN3270 access to the host, and then add the service account to the Configured Users folder. Among other things, this provides encryption if the TN3270 service uses LUs on another server.

- Enable LUA Security for TN5250 access to the host. This increases security by requiring explicit assignment of LUs to user records.

- When using the printing feature associated with the TN3270 Server, reconfigure the display and printer to use the same port. This is necessary because, since these two items are configured separately, they are often inadvertently configured to different ports, and subsequently different security settings.

- Always use IPSec when using the TN3270 or TN5250 Servers. Although data might be safe between the client and server without IPSec, that same data may become vulnerable between the server and the host. Using IPSec reduces the attack surface, ensures data encryption, and makes access available only to authorized users.

See Also

Other Resources

Understanding Windows Security

Resource Security

This section describes accessing Host Integration Server resources, including Display LUs, Printer LUs, and LU pools.

In This Section

3270 Terminal Access

Users or groups who require access to 3270 sessions from workstations using Host Integration Server client applications must be members of the SNA subdomain. Therefore, they are also members of the Windows domain of which the subdomain is a part.

Once enrolled in the SNA subdomain, you can assign specific 3270 (LU type 2) resources to the appropriate accounts. The users can access only the specific resources you allocate to them.

The preferred method for maintaining security in your environment is by using domain security to authenticate users, and then limiting their access by assigning them only specified resources.

See Also

Other Resources

Resource Security

5250 Terminal Access and APPC Access

Users who want Advanced Program-to-Program Communications (APPC) access need not be defined in the SNA subdomain, but they must be members of a Windows domain.

For 5250 terminal access using a Host Integration Server client computer within the network, the AS/400 supplies the required security for logon to the AS/400.

For APPC access programmed into specific applications, you maintain security through the actual programmatic conversation, if required.

See Also

Other Resources

Resource Security

Securing the TN3270 and TN5250 Services

You can help secure TN3270 and TN5250 services by specifying client workstation IP addresses that have permission to use the resources provided. With TN3270E clients, you can specify a workstation name in place of the client IP address.

The method employed for verifying workstations can also be used to allow only specified IPs to request resources allocated to them.

See Also

Other Resources

Resource Security

Transaction Integrator Security

Security affects Transaction Integrator (TI) in two ways. First, TI components can be assigned security attributes in the same way as other COM+ components. This requires no TI development. Second, the TI run-time environment needs to deal with the security mechanisms of the remote environment (RE). TI provides two security options with an optional override for each:

- Package-level (also known as application-level)
- 4User-level
- Optional explicit-level override

When configured for user-level credentials, TI makes use of the APPC Privileged Proxy feature for single sign on. This requires that the user context that the APPC application (TI, in this case) is running under be a member of the **HSDomain_Proxy** group. (The **HSDomain_Proxy** group is one of the two groups created when the host security domain is created.) By default, the **HSDomain_Proxy** group contains the **Domain Admins** group. If TI is not running under the context of a user in the **Domain Admins** group, you will need to add the user to the **HSDomain_Proxy** group.

When deploying a TI component, the administrator must choose either package-level security or user-level security as the default. The optional explicit-level security override is a separate option that the administrator can enable or disable; the override applies regardless of which security option (package-level or user-level) is in place. If the explicit-level override is disabled, base applications will not be permitted to use the callback to provide user credentials. The administrator can also turn on the optional Already Verified settings.

In This Section

Application-Level Security

When using application-level (or package-level) security, the Transaction Integrator (TI) run-time environment uses the Single Sign-On support in Host Integration Server to authenticate itself to the host by using the Windows 2000 security identity of the COM+ application. The COM+ application containing the TI component must be configured in COM+ to run under a Windows account that maps to valid mainframe credentials.

When a TI component is deployed in a COM+ Application which itself is configured as a Library application or package (instead of a Server application or package) on the **Activation** tab of its property page, the choice of security level (Package-level or User-level) for a TI remote environment (RE) is ignored. A Library application (or package) runs in the Windows user credentials of the process that invoked it, and TI attempts to map this name to a corresponding host user ID.

Although the TI security level is ignored in Library COM+ applications, both of the following security options on the RE property page are accepted by both Library and Server COM+ applications:

- Allow application to override selected authentication
- Use Already Verified or Persistent Verification authentication

The Library application deployment model is supported by COM+. It offers better performance than the Server application deployment model in certain situations. TI supports both models. For example, Internet Information Services (IIS) documentation recommends that COM+ Automation servers that will be invoked from an active server page (ASP) be deployed in a COM+ Library application rather than a COM+ Server application. This way, IIS can control the level of multitasking on a system and kick off new Windows processes to support very high hit rates. By deploying in a Library Application, each IIS process gets its own set of TI components. Also, as a developer debugging a new application, you might prefer to debug a Library COM+ application instead of a Server COM+ application because the Library application simplifies debugging. All the breakpoints come up in one Windows process, and you can more easily see the flow of control.

To set the security level

1. Start TI Manager, and double-click **COM Transaction Integrator** for the computer.
2. Double-click **Remote Environments**.
3. Right-click the specific remote environment (RE) that you want to set, and then click the **Security** tab.
4. Select the **Set security on** check box.
5. Click **Authenticate with package credentials**, or click **Authenticate with User credentials**.

See Also
Other Resources
Transaction Integrator Security

User-Level Security

If user-level security is enabled, the TI run-time environment provides the client identity to the Single Sign-On support of Host Integration Server before allocating the LU 6.2 conversation or connecting to the remote TCP/IP end point. This enables TI to use the client's identity (the identity of the client that invoked the COM+ component that resulted in the invocation of the TI component) to map to a mainframe credential by using the Host Integration Server Single Sign-On capability to authenticate itself with the mainframe.

When a TI component is deployed in a COM+ Application which itself is configured as a Library application or package (instead of a Server application or package) on the **Activation** tab of its property page, the choice of security level (Package-level or User-level) for a TI remote environment (RE) is ignored. A Library application (or package) runs in the Windows user credentials of the process that invoked it, and TI attempts to map this name to a corresponding host user ID.

Although the TI security level is ignored in Library COM+ applications, both of the following security options on the RE property page are accepted by both Library and Server COM+ applications:

- Allow application to override selected authentication
- Use Already Verified or Persistent Verification authentication

The Library application deployment model is supported by COM+. It offers better performance than the Server application deployment model in certain situations. TI supports both models. For example, Internet Information Services (IIS) documentation recommends that COM+ Automation servers that will be invoked from an active server page (ASP) be deployed in a COM+ Library application rather than a COM+ Server application. This way, IIS can control the level of multitasking on a system and kick off new Windows processes to support very high hit rates. By deploying in a Library Application, each IIS process gets its own set of TI components. Also, as a developer debugging a new application, you might prefer to debug a Library COM+ application instead of a Server COM+ application because the Library application simplifies debugging. All the breakpoints come up in one Windows process, and you can more easily see the flow of control.

See Also

Other Resources

Transaction Integrator Security

Single Sign-On in Transaction Integrator

Single Sign-On (SSO) is a mechanism that enables a user to enter a user name and password once to access multiple applications. It gives users simplified logon and maintenance of the many passwords needed to access Windows and back end systems and/or applications. It enables applications to integrate by automating the process of logging on to the host/backend system.

In general, there are three types of single sign-on services:

- **Common authentication single sign-on:** These services enable you to connect to multiple applications within your computer. Your credentials are requested and verified once when you log into the computer, and these credentials are used to determine what actions you can perform based on your permissions. An example of this type of single sign-on is Exchange Server, which uses Windows Integrated Security. Once the user has logged on to Windows, they do not need to provide additional credentials to access their e-mail.

- **Internet single sign-on:** These services enable you to access resources over the Internet by using a single set of user credentials. The user provides a set of credentials to log on to different Web sites that belong to different organizations. An example of this type of single sign-on is Windows Live ID.

- **Intranet single sign-on:** These services enable you to connect to multiple heterogeneous applications and systems within the enterprise environment in your company. An example of this type of single sign-on is Enterprise Single Sign-On, which provides a framework for Windows applications to integrate with non-Windows applications to enable single sign-on.

SSO facilitates the translation of credentials by the Security Policy Wizard. SSO lets you define and manage the relationship between foreign host user ID and password credentials to Windows credentials. The basis for managing these relationships is the SSO Affiliated Application.

An Affiliated Application is a grouping of well defined host user identities under a logical entity that reflects the host administrators view of application processing systems in the non-Windows environments.

When SSO is handed a set of host credentials (user ID and password) along with a valid SSO Affiliated Application, SSO returns a Windows Access Token that represents the Windows credentials. HIP uses this Access Token when setting up the execution thread for methods on server object.

To make this process easier, the Security Policy must know which SSO Affiliated Applications are to be queried in an attempt to gain access to the Windows credentials (access token) needed to execute methods on the server object. The wizard pane lets the user select from a list of valid SSO Affiliated Applications and add them to the Security Policy being defined. The wizard pane also lets the user remove SSO Affiliated Applications previously defined to the Security Policy.

Special Security Settings for TCP/IP

Two special security settings are available for the TCP/IP protocol for CICS and IMS:

- Link and Concurrent Server Mainframe authentication for the TCP/IP CICS MS Link and Concurrent server must have a user exit, EZACIC06, coded on the mainframe to validate the user ID and password supplied by TI. The user ID and password are part of the CICS transaction request message that is formatted by TI and used to initiate the TCP/IP server application in CICS. Refer to the IBM TCP/IP and CICS documentation for further detail.

- For TCP/IP IMS Explicit, IMS Implicit, and IMS Connect or OTMA Mainframe authentication for TCP/IP IMS Explicit, IMS Implicit, and IMS Connect or OTMA must have a user exit, IMSLSECX, coded on the mainframe to validate the user ID and password supplied by TI. The user ID and password are part of the IMS transaction request message that is formatted by TI and used to initiate the TCP/IP server application in IMS. Refer to the IBM TCP/IP and IMS documentation for further detail.

See Also
Other Resources
Transaction Integrator Security

Mainframe Authentication for CICS LINK

Resource-level authentication is recommended in the CICS region. Due to a restriction imposed by the IBM Distributed Program Link (DPL) protocol, a user ID and password transmitted from the workstation by TI are ignored and not used for transaction-level authentication. The target CICS region expects, under the circumstances, that authentication has been completed by the application that executes the IBM DPL; for example, a TI application on the PC. (Traditionally, the application that executes an IBM DPL has been another CICS region.) Instead, for transaction-level authentication, the target CICS region associates the default user ID for the region with the transaction ID of the CICS (mirror transaction) task and the user ID from the sender is ignored. Unless this is taken into consideration, attempts to secure the mirror transaction can cause an application malfunction because of the failure to authenticate.

See Also

Other Resources

Transaction Integrator Security

AS/400 Security

The support for AS/400 security is the same as for other Windows-initiated operations against the mainframe, with the following adjustments:

- No support for RACF, AFC/2, Kerberos, or Top Secret.
- Integrate with AS/400 native security system only.
- Support Single Sign-On via SSO.

See Also

Other Resources

Transaction Integrator Security

Limitations of User Access Level Sign On

When you sign on with only user access permissions, you have restricted capabilities for using Transaction Integrator. For TI Manager, user access:

- Allows you to view COM+ applications
- Does not allow you to open TI Manager
- Does not allow you to export a COM+ application.

For TI Designer, user access:

- Allows you to open TI Designer
- Allows you to create and save new type libraries
- Allows you to open and save existing type libraries.

See Also

Other Resources

Transaction Integrator Security

Transaction Integrator Threat Mitigation

Product security is a top priority across Microsoft. Beginning with the Windows Security Push in 2002, Microsoft has invested additional time and resources to developing more secure code and detailed instructions for deploying and securing your computing environment. The Host Integration Server product team conducted a complete threat modeling analysis to identify and mitigate potential areas of concern. A threat model is a security-based analysis that helps you determine the highest-level security risks posed to a product or application and how attacks can manifest themselves.

Although Microsoft has mitigated all possible internal security threats to Host Integration Server, there are steps you should take to mitigate threats from elsewhere in your network environment. Threat modeling helps you evaluate the threats to the applications you are writing or running, and thereby reduce the overall risk to your computer system. For more information about threat model analysis, see Chapter 4 Threat Modeling in Michael Howard and David LeBlanc, *Writing Secure Code 2nd Edition*, Redmond, WA: Microsoft Press. 2003.

Howard and LeBlanc summarize six categories of possible security threats to your computing environment:

- **Spoofing identity**. Spoofing threats allow an attacker to pose as another user or allow a rogue server to pose as a valid server. An example of user identity spoofing is illegally accessing and then using another users authentication information, such as username and password.

- **Tampering with data**. Data tampering involves malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.

- **Repudiation**. Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performing an illegal operation in a system that lacks the ability to trace the prohibited operations.

- **Information disclosure**. Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, a users ability to read a file that she was not granted access to and an intruders ability to read data in transit between two computers.

- **Denial of service**. Denial of service (DoS) attacks deny service to valid users— for example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.

- **Elevation of privileges**. In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.

Howard and LeBlanc also point out that some threat types can interrelate. For example, it is possible for information disclosure threats to lead to spoofing threats if the users credentials are not secured. Similarly, elevation of privilege threats are by far the worst because if someone can become an administrator or root on the target computer, every other threat category becomes a reality. Conversely, spoofing threats might lead to a situation where escalation is no longer needed for an attacker to achieve his goal.

To mitigate threats that originate outside Transaction Integrator but which can negatively affect TI components and your application, Microsoft recommends that you do the following:

- Protect the MSHIS60_HIP Database and SQL Server Stored Procedures

- Protect the TI DCGen COM+ Application and TCP/IP Listeners

- Protect the COM+ and .NET Servers

- Protect the Raw User Data

- Protect the HIP Listener
- Protect the Local File System, Database, and Registry
- Protect the Client Proxy
- Protect the Remoting Session
- Protect TI from Unauthorized Mainframe or AS/400 Access
- Protect the TI Runtime Environment
- Protect Mainframe Security Credentials from Being Overridden
- Protect the TI Runtime and Host Environments from Data Overflows

Protect the MSHIS60_HIP Database and SQL Server Stored Procedures

To prevent an attacker from spoofing their identity or tampering with the data or stored procedures in the HIP database, you should do the following:

- Run the HIP Service in a valid Windows privileged account and use Windows integrated security to access the HIP SQL Server database.
- Allow only privileged accounts access to the HIP SQL Server database.
- Use a local SQL Server database and do not allow remote access.
- Use only Windows integrated security and restrict access to only privileged Windows accounts.

You can also help mitigate this threat with the following deployment scenarios:

- Service-based Security
- Host-initiated Single Sign-On (SSO).

You can learn more about this threat by reading about:

- How to secure remote access to SQL Server (for example, integrated Windows security).

Protect the TI DCGen COM+ Application and TCP/IP Listeners

To prevent an attacker from spoofing their identity, tampering with the data on the host, elevating their privileges, or denying service, you should:

- Use ASP.NET Admin to set the level of user or role-based security for Application Directory
- Tighten COM+ Application Security on the TI DCGen package
- Use TI Manager to set the security level for a remote environment
- Move the TI Designer-generated .dll to the remoting client computer.

You can also help mitigate this threat with the following deployment scenario:

- ASP.NET web client

You can learn more about this threat by reading about:

- ASP.NET application and IIS security
- Multiple levels of authorized users.

See Also
Reference
New Security Policy Wizard

Protect the COM+ and .NET Servers

To prevent an attacker from spoofing their identity, tampering with the data on the host, elevating their privileges, accessing restricted data, or denying service, you should:

- Use bounded recordsets
- Use static IP addresses on the host
- Run server components in-process with a HIP Service account.

You can also help mitigate this threat with the following deployment scenario:

- Send properly-formatted messages (for example, ELM, TRM, DPL, SNA, IP)
- Service-based security

See Also
Concepts
Bounded Recordsets

Protect the Raw User Data

To prevent an attacker from reading data packets on the network and either tampering with the data or disclosing the data, or to prevent an attacker from denying service, you should:

- Place the computer running Transaction Integrator (TI) and the host in a secure location.
- Use direct network connections over a private segment
- Use IPSec, where supported, for IP communications between host and Windows servers
- For SNA LU6.2 network connections to mainframe or AS/400 computers, use Host Integration Server server-to-server data encryption when deploying upstream an Host Integration Server computer as a SNA gateway to a downstream TI HIP computer
- For SNA LU6.2 network connections to mainframe, use Host Integration Server IP-DLC Link Service in conjunction with IPSec.

See Also

Reference

Configuring an IP-DLC Link Service

Protect the HIP Listener

To prevent an attacker from spoofing their identity, tampering with the data, or denying service, you should run the HIP services in valid Windows privileged account.

To protect the HIP listener

1. Click **Start**, click **Control Panel**, double-click **Administrative Tools**, and then double-click **Services**.
2. In the details pane, right-click **HIPService**, and then click **Properties**.
3. Click **This account**,and then specify a valid Windows privileged account.
4. Type the password for the user account in **Password** and in **Confirm password**, and then click **OK**. If you select the **Network Service** account, the password must be blank.

You can also help mitigate this threat with the following deployment scenario:

- Service-based security

See Also
Reference
Services Account Screen

Protect the Local File System, Database, and Registry

To prevent an attacker from spoofing their identity, accessing restricted data, or tampering with the data, you should do the following:

- Place the computer that is running Transaction Integrator (TI) in a secure location.
- Confirm that the access permissions to TI programs, TI components, and the registry are set correctly.
- Use host-initiated Single Sign-On (SSO) in conjunction with valid host UID and PWD passed in the initial connection flows. You should not run the COM Server program in context of the default COM+ security context. Using both UID and PWD causes the HIP application to validate both items when calling SSO.
- Not use DCOM to remote calls from HIP Application server to a second Windows Server hosting COM+ server application.
- Use a secure network connection (for example, CICS TRM over IPSec-protected TCP/IP network connection).

You can also help mitigate this threat with the following deployment scenarios:

- Service-based security
- Host-initiated SSO

You can learn more about this threat by reading about the following:

- How to help secure remote access to SQL Server (for example, integrated Windows security)
- How to send a valid host user ID
- How to send a valid host password

See Also

Reference

New Security Policy Wizard

Concepts

Single Sign-On in Transaction Integrator

Protect the Client Proxy

To prevent an attacker from tampering with a properly formatted message (for example, a TRM, DPL, SNA, or IP) sent by the client computer to the TI runtime or to prevent an attacker from sending a malformed message to the TI runtime, you should do the following:

- Move the .NET Framework assembly DLL generated by TI Designer to the remote client computer.

You can learn more about this threat by reading about the following:

- .NET remoting
- ASP.NET Web clients

See Also

Other Resources

Transaction Integrator Threat Mitigation

Protect the Remoting Session

To prevent an attacker from tampering with the remote session or denying service, you should do the following:

- Use HTTPS between the remoting client and the server
- Use separate ASP.NET application contexts for various levels of privileged users, thereby preventing lower-privileged users from affecting sessions belonging to higher-privileged users.

You can also help mitigate this threat with the following deployment scenario:

- ASP.NET Web client.

You can learn more about this threat by reading about the following:

- Multiple levels of authorized users
- ASP.NET application and IIS security
- Properly-formatted TRM, DPL, SNA, IP, and other messages
- Valid host user ID and password
- Valid Windows user ID for Single Sign-On (SSO).

See Also

Other Resources

Transaction Integrator Threat Mitigation

Protect TI from Unauthorized Mainframe or AS/400 Access

To prevent an attacker from using the mainframe or AS/400 connection to access the server running Transaction Integrator, you should:

- Utilize direct network connections over private segment.
- Use IPSec, where supported, for IP communications between the host and Windows servers.
- For SNA LU6.2 network connections to mainframe or AS/400 computers, use Host Integration Server server-to-server data encryption when deploying upstream Host Integration Server computer as SNA gateway to downstream TI HIP computer.
- For SNA LU6.2 network connections to mainframe, use Host Integration Server IP-DLC Link Service in conjunction with IPSec.

See Also
Concepts
Secure Deployment of the IP-DLC Link Service
Other Resources
Introduction to the IP-DLC Link Service

Protect the TI Runtime Environment

To prevent an attacker from instantiating multiple Transaction Integrator (TI) components to launch a denial of service attack on the TI runtime environment, you should:

- Store all TI component type libraries and .NET assemblies in a secure directory.
- Confirm that the access permissions on all type libraries and .NET assemblies are set correctly.
- Confirm that the access permissions on the directory where the type libraries and .NET assemblies are stored are set correctly.

See Also

Other Resources

Transaction Integrator Threat Mitigation

Protect Mainframe Security Credentials from Being Overridden

To prevent an attacker from gaining control over security credentials used to access a mainframe host, you should do the following:

- Use host-initiated Single Sign-On (SSO) in conjunction with valid host UID and PWD passed in the initial connection flows.

- Set the ClientContext to not allow security credentials to be overridden when using the **SelectionHint** property.

See Also

Tasks

Specifying a Remote Environment Programmatically

Other Resources

Remote Environment Selection Using the SelectionHint Property

Protect the TI Runtime and Host Environments from Data Overflows

To prevent an attacker from using inbound, unbounded recordsets to launch a denial of service attack on either the Transaction Integrator (TI) runtime or host environment, you should:

- Store all TI component type libraries and .NET assemblies in a secure directory.
- Confirm that the access permissions on all type libraries and .NET assemblies are set correctly.
- Confirm that the access permissions on the directory where the type libraries and .NET assemblies are stored are set correctly.

See Also

Concepts

Unbounded Recordsets

Deployment

This section provides instructions for deploying and configuring Host Integration Server. This includes how to test and verify connections with the mainframe or AS/400 computer.

In This Section

Installation Guide

This document contains information about the basic installation process for Microsoft® Host Integration Server 2009.

You can download the latest version of this topic at http://go.microsoft.com/fwlink/?LinkId=141188.

It is important to read the entire document before you start the installation because it contains hardware and software requirements, correct installation procedures, and other information necessary for successful installation.

Note
It is also important to read the Readme.htm file before you start installation. The Readme.htm file is located in your download folder, and contains late-breaking information and known issues that you should understand before installation.

This document contains:

- Hardware and Software Requirements
- Installing the Product

Hardware and Software Requirements

In the following sections, the version numbers listed are minimum requirements. Versions later than those listed here are supported unless otherwise noted.

Hardware Prerequisites

The minimum hardware requirements for a complete installation of Host Integration Server 2009 are as follows:

- 450 megahertz (MHz) or higher Intel Pentium-compatible CPU
- 512 megabytes (MB) of RAM
- 6-gigabyte (GB) hard disk
- CD drive or DVD drive
- Super VGA monitor (800 x 600) or higher-resolution monitor with 256 color
- For HIS Designer only: Display Super VGA (1024x768) or higher-resolution monitor with 256 colors
- Microsoft Mouse or compatible pointing device

Software Prerequisites

Important
Microsoft Host Integration Server 2009 will not install on a computer that already has the Microsoft OLE DB Provider for DB2 installed. If you have the OLE DB provider installed on your computer, you must uninstall it before you install Host Integration Server 2009.

Supported Operating Systems

To install Host Integration Server 2009, you must be running one of the following operating systems:

- Microsoft Windows® XP Professional with Service Pack 2 (x86 or x64)
- Microsoft Windows Server® 2003 Service Pack 2 (x86 or x64)
- Windows Server 2003 R2 Service Pack 2 (x86 or x64)

- Microsoft Windows Vista® Service Pack 1 (x86 & x64)
- Windows Server 2008 (x86 & x64)

Product Prerequisites

Host Integration Server 2009 requires the following prerequisites:

- .NET Framework version 3.5 SP1
- Microsoft XML Core Services (MSXML) 6.0 SP1

Note
Setup will install redistributable prerequisites automatically from the Web or from a pre-downloaded CAB file. Automatically installing from the Web or downloading the CAB file requires Internet access. You can download the CAB file at: http://go.microsoft.com/fwlink/?LinkId=133541.

Feature Prerequisites

If you choose to install only selected features of Host Integration Server 2009, the following tables outline which software prerequisites you will need.

Server Installation

Feature	Requirement
Developer Design Tools	Microsoft Visual Studio® 2005 SP1 or Visual Studio 2008
BizTalk Adapter for Host Applications	Microsoft BizTalk® Server 2006, BizTalk Server 2006 R2, or BizTalk Server 2009
BizTalk Adapter for DB2	BizTalk Server 2006, BizTalk Server 2006 R2, or BizTalk Server 2009
BizTalk Adapter for Host Files	BizTalk Server 2006, BizTalk Server 2006 R2, or BizTalk Server 2009
BizTalk Adapter for WebSphere MQ	BizTalk Server 2006, BizTalk Server 2006 R2, or BizTalk Server 2009
MSMQ-MQSeries Bridge	Message Queuing with routing support

Configuration After Server Setup

You can install the Server software without running the Configuration Tool, and then run the Configuration Tool later (on the **Start** menu). Eventually, however, you must run the Configuration Tool. To do this, you will need the software listed in the following table. It can be installed either on the local Host Integration Server computer or on a remote computer.

Feature	Requirement
DB2 Distributed Transactions	Microsoft SQL Server® 2005 SP2 or SQL Server 2008
Transaction Integrator	SQL Server 2005 SP2 or SQL Server 2008
Enterprise Single Sign-On	SQL Server 2005 SP2 or SQL Server 2008

MQ Connectors

The following are the supported versions of WebSphere MQ:

Software Requirements

You must have the following software installed before you can install the adapter.

- BizTalk Adapter for WebSphere MQ (Client-Based)
    - IBM WebSphere MQ Client 6.0 with Fix Pack 6.0.1.1 or higher

    - IBM WebSphere MQ Client 7.0 (required for 64-bit)
    - IBM WebSphere MQ Extended Transactional Client (for transactional support)
- MSMQ-MQSeries Bridge
    - IBM WebSphere MQ Client 6.0 with Fix Pack 6.0.1.1 or higher
    - IBM WebSphere MQ Client 7.0 (required for 64-bit)
    - IBM WebSphere MQ Extended Transactional Client (for transactional support)
- WCF Channel for WebSphere MQ
    - IBM WebSphere MQ Client 6.0 with Fix Pack 6.0.1.1 or higher
    - IBM WebSphere MQ Client 7.0 (required for 64-bit)
    - IBM WebSphere MQ Extended Transactional Client (for transactional support)

Note
When using Extended-Client, in addition to configuring the adapter receive locations and send port, ensure that you refer to t he “How to Configure the MQSC Adapter for Transactional Messaging” topic in the Host Integration Server 2009 documentat ion for additional information. Also refer to the Known Issues section in the Readme file.

Refer to your IBM documentation for additional information about how to obtain and install WebSphere MQ Extended Transactional Client or WebSphere MQ Client.

Installing the Product

The following sections contain installation information for Host Integration Server 2009.

Important Notes About Installation

If you are upgrading a Host Integration Server 2004 or Host Integration Server 2006 computer, Host Integration Server 2009 Setup will upgrade your configuration automatically.

New to the Product Installation

The product installation now includes both Server and Client components, and also a developer-only and documentation-only installation.

To Install Host Integration Server 2009

1. In Windows Explorer, locate the installation folder in which you extracted the Setup files.
2. Double-click **Setup.exe**. The Host Integration Server 2009 Installation panel appears.
3. Click the product name to start installation.

   The Installation Wizard appears and guides you through the rest of the installation process.
4. Click **Finish** when you are finished.

To Uninstall Host Integration Server 2009 Using Add or Remove Programs

1. Click **Start**, point to **Settings**, and then click **Control Panel**.

2. In **Control Panel**, open **Add or Remove Programs**, select the name of the product, and then click **Change/Remove**.

   The **Add or Remove Application** dialog box appears.

3. Click **Remove**, and then click **Next**.

   The Uninstall Wizard appears and automatically starts the uninstall process.

4. When the wizard has completed the process, click **Finish**.

Note
Uninstalling Host Integration Server 2009 does not uninstall Enterprise Single Sign-On (SSO). To uninstall SSO, see "Uninstalling Enterprise Single Sign-On" later in this document.

Configuring Host Integration Server 2009

Host Integration Server 2009 contains an update to the Configuration Tool that is used during setup to complete the initial installation. On first use of the Configuration Tool, a **Configuration – Start** panel gives the user the option of doing a Basic or Custom configuration.

Basic Configuration

To install by using the default setting, you can use the **Basic configuration** option. When you click **Configure**, the tool proceeds using the default settings, together with the minimum information supplied by the user on the **Configuration - Start** panel.

Custom Configuration

If you choose **Custom configuration** and then click **Configure**, you are presented with a list of configuration panels based on the features selected during installation. Selecting an item in the left pane displays its associated settings in the right pane. After you have entered your configuration settings, you can click **Apply Configuration** in the toolbar to implement those settings. Note that any settings made in the initial **Configuration - Start** panels are populated as defaults in the custom configuration.

Updating the Configuration

You can update the configuration later by running the Configuration Tool on the **Start** menu. Individual settings can be changed on each panel. To update the service account or database settings for all panels, click the **View** menu and then select **Service Accounts** or **Databases**. In these dialog boxes you can multi-select all the rows by using the CTRL key, and bulk edit the settings.

Unconfigure

A feature can be disabled by unconfiguring that feature. Select **Unconfigure Feature** on the menu, select the feature to be unconfigured, click **OK**, and then continue through the wizard. This process removes any settings (for example, services and registry keys) that were created when the configuration was previously applied.

Unattended Configuration

The Configuration Tool can export a configuration file that can be used to perform an unattended installation and configuration. Select **Export Configuration** on the **File** menu. The unattended installation consists of two steps: setup and configuration.

Unattended Installation

Unattended installation is useful for installing Host Integration Server 2009 on a large number of computers to ensure that they are all configured in exactly the same way. It is also useful if you need to change or update the existing configuration of a large number of computers.

You perform an unattended installation by creating a "model" installation on one computer, which you then apply to the other computers.

To perform unattended installations

1. Using the standard Windows-based installation program, perform a full installation on a single computer. Because the installation parameters you choose will be saved as a model, make the selections you want to use on the other installations.

2. When setup is complete, run the Configuration Tool on the same computer. As in step 1, make the selections you want to use on the other installations. Select **Export Configuration** on the **File** menu, and save the configuration XML file.

3. If you need to update the configuration XML file, select **Import Configuration**, open a saved file, and make any necessary changes. This option is available only until you have completed configuration. If you need to make an update after configuration, you will need to configure the system again.

4. On the appropriate server computer, open a command prompt and run the Unattended Installation command (see the following sections). Where the command calls for the configuration file, use the name and location of the HISServerConfig.xml file created in step 1. You can also use the options listed in the "Options" section.

Commands

Use the following commands and options for unattended installation.

```
Setup.exe /InstallPlatform /L c:\HISInstall.log /S c:\HISServerConfig.xml /INSTALLDIR C:\HI
S
```

```
Configuration.exe /L c:\ConfigFramework.Log /S c:\HISServerConfig.xml
```

Options

/InstallPlatform

This flag causes the installation program to install any platform prerequisites.

/L c:\HISInstall.log

This flag determines the log file location that is created during setup.

/S C:\HISServerConfig.xml

This flag specifies the configuration file (list of features) that is used on installation. The file "HISServerConfig.xml" contains information about which features to install and how they should be configured. Sample copies of these files are located in the Support\Unattended_Installs directory of the installation folder. This file can also be created during setup by clicking **Save** in the summary panel of the Configuration Tool.

/INSTALLDIR

This option tells Setup where to install the product.

Unattended Uninstall

Use the following command for unattended uninstall:

```
Setup.exe /l C:\UninstallLog.txt /REMOVE ALL /Product HIS
```

Note that Setup.exe will need to have the install location appended to the beginning to ensure that the correct Setup.exe program is launched.

Uninstalling BizTalk Adapters

Uninstall will not delete the adapter registration from BizTalk Server. This is because there could be other instances of the adapter installed on different BizTalk Server computers in the group. If you want to clean this up, after you have uninstalled the last adapter installation from all the BizTalk Server computers in the group, you can manually delete the adapter from the BizTalk Server Administration console or by using Windows Management Instrumentation (WMI).

Uninstall will not undeploy the schema DLLs (for example, MQSeriesEx.dll or MQSeries.dll) from the BizTalk Management database.

Uninstalling Enterprise Single Sign-On

Uninstall

Uninstalling Host Integration Server does not uninstall Enterprise Single Sign-On. To uninstall this feature, follow these steps:

1. Click **Start**, point to **Settings**, and then click **Control Panel**.

2. Double-click **Add or Remove Programs**.

3. In the **Add or Remove Programs** dialog box, click **Microsoft Enterprise Single Sign-On**, and then click **Remove**.

4. Click **Yes** when you are prompted to confirm removal of Microsoft Enterprise SSO.

Note
If you have BizTalk Server Runtime, Development, or Administration features installed, or Host Integration Server Administration features installed, you cannot uninstall the SSO Runtime or Administration components until all dependencies are removed.

Unattended Uninstall

Note
You will need the original installation media to perform unattended uninstalls of Single Sign-On Server or Client versions.

Use the following commands for unattended uninstall:

Host Integration Server 2009 Server

```
MSIEXEC /X <Drive>\Platform\SSO\SSO.msi\SSO.MSI
```

Host Integration Server 2009 Client

```
MSIEXEC /X <Drive>\Platform\SSO\CLIENT\SSOClient.msi
```

Note
If you have BizTalk Server Runtime, Development, or Administration features installed, or Host Integration Server Administration features installed, you cannot uninstall the SSO Runtime or Administration components until all dependencies are removed.

Special Considerations When Installing Enterprise Single Sign-On

The following sections contain information about installation of the Enterprise Single Sign-On (SSO) feature. Because of this feature's complex relationships to other features and systems, and because of its importance to system security, we recommend that you read these sections carefully before you install Enterprise Single Sign-On.

Installing SSO and Creating the Master Secret Server

Initial installation of Enterprise Single Sign-On must be done on the server that you will use as the Master Secret Server. This is also the only Master Secret Server allowed in the entire SSO system.

To install Enterprise Single Sign-On, run the Host Integration Server 2009 or BizTalk Adapters for Host Systems Setup program and under the **Server** node, select **Enterprise Single Sign-On** and continue installation.

After installation is complete, the Configuration Wizard starts. Select the Custom Configuration mode. Because this is the Master Secret Server, select the option to "Create a new SSO System" when configuring Enterprise SSO. This creates the Master Secret Server and the SSO Credential database.

Processing Servers for Enterprise Single Sign-On

In a multi-computer environment, after the Master Secret Server and Credential database have been created, you can install Enterprise Single Sign-On on subsequent computers. These are typically the computers on which Host Integration Server 2009 is also installed.

The initial installation process is the same as on the first computer. Configuration, however, becomes slightly different. Because the Master Secret Server and the SSO Credential database are already in place, select "Join an existing SSO System" when configuring Enterprise SSO.

EntSSO Administration Installation Only

You can install just the Enterprise Single Sign-On Administration feature. To do this, when you run the installation program, under the **Client** node, select only **Enterprise Single Sign-On Administration** and continue installation.

Only SSO administrative components will be installed. The hardware and software requirements are the same as for a typical EntSSO installation.

Note
While the SSO Administration feature in Host Integration Server 2009 is compatible with the server version of SSO in BizTalk Server 2009, the administrative components of Enterprise SSO in BizTalk Server 2009 are not compatible with the server version of Enterprise SSO in Host Integration Server 2009. Enterprise SSO MMC snap-ins require MMC 3.0. It is not supported on Windows 2000.

Installing the SSO Administration feature installs command-line-based utilities (ssomanage.exe and ssoconfig.exe) and an MMC snap-in that can be accessed by clicking **Start**, pointing to **All Programs**, and then clicking **Microsoft Enterprise Single Sign-on**. To run the SSO administrative command-line utilities after installation, you must open a command prompt and navigate to the SSO directory located at Program Files\Common Files\Enterprise Single Sign-On. You can use ssomanage.exe to specify the SSO server you want to use for management. To do this, open a command prompt and navigate to the SSO directory located at Program Files\Common Files\Enterprise Single Sign-On, and then run **ssomanage -serverall <SSO server name>**.

SSO Client Installation

The SSO client utility (ssoclient.exe or SSOClientUI.exe) enables end users to configure their client mappings in the Credential database. You can install the client utility from a self-extracting file (SSOClientInstall.exe), which is installed with the SSO Administration feature. This also installs the UI-based SSO client utility that can be accessed by clicking **Start**, pointing to **All Programs**, and then clicking **Microsoft Enterprise Single Sign-on**.

To install the SSO client utility, you must be running one of the following operating systems on the client computer:

- Windows Server 2003 (x86 or x64), Windows Server 2003 R2 (x86 or x64)
- Windows XP Professional with Service Pack 1
- Windows Vista

Note
The UI-based SSO client utility (SSOClientUI.exe) requires .NET Framework 2.0.

To run the command-line-based SSO client utility (ssoclient.exe) after installation, you must open a command prompt and navigate to the SSO directory located at Program Files\Common Files\Enterprise Single Sign-On.

Password Synchronization

To install the password synchronization feature, run the installation program under **Server – Enterprise Single Sign-On**; select the **Enterprise Single Sign-On Password Management** sub-feature and continue with installation.

Password Change Notification Service (PCNS) can be obtained from the following location: http://go.microsoft.com/fwlink/?LinkID=68145.

Upgrading from an Earlier Version of SSO

Host Integration Server 2009 and the Microsoft BizTalk Adapters for Host Systems include Microsoft Enterprise Single Sign-On version 4.0. Previous Microsoft products include the following versions of Enterprise Single Sign-On:

- Enterprise SSO v1 is included in Microsoft BizTalk Server 2004
- Enterprise SSO v2 is included in Microsoft Host Integration Server 2004
- Enterprise SSO v3 is included in Microsoft BizTalk Server 2009 and Microsoft Connected Services Framework Server 3.0

Upgrade Procedure

If Enterprise SSO was installed with Host Integration Server 2004, installing the BizTalk Adapters for Host Systems or Host Integration Server 2009 will automatically upgrade SSO. (Note that you must first perform the upgrade on the Master Secret Server.)

If Enterprise SSO was installed with one of the preceding products other than Host Integration Server, follow the procedure below to upgrade to this release of SSO.

To perform these operations, you must be an SSO Administrator and a computer Administrator.

1. Make a secure backup of the SSO Credential database from your SQL Server computer before performing an upgrade.

2. Make sure you have a secure backup of the current secret from the Master Secret Server.

3. Upgrade Enterprise SSO on the Master Secret Server. To do this, follow these steps:

    - Run **setup.exe** for Microsoft BizTalk Adapters for Host Systems or Microsoft Host Integration Server 2009 package.

    - In the custom setup tree, select **Enterprise Single Sign-On**, and clear all other options to only install Enterprise SSO. This will uninstall the earlier version and install the new version.

    - After installation is completed, start the Configuration Wizard to configure Enterprise SSO.

    - In the Configuration Wizard, select **Custom Configuration**. In the left pane, select **Enterprise SSO**, and in the right pane, select **Enable Enterprise Single Sign-On**. All the settings will be unavailable. Select **Apply Configuration**, and then click **Next** to continue with the configuration. This will also upgrade the SSO database if it is required.

    - After configuration is completed, restore the secret from the backup on the Master Secret Server.

4. Upgrade Enterprise SSO on other SSO servers. To do this, follow these steps:

    - Run **setup.exe** for Microsoft BizTalk Adapters for Host Systems or Microsoft Host Integration Server 2009 package.

    - In the custom setup tree, select **Enterprise Single Sign-On**, and clear all other options to only install Enterprise SSO. This will uninstall the earlier version and install the new version.

    - After installation is completed, start the Configuration Wizard to configure Enterprise SSO.

    - In the Configuration Wizard, select **Custom Configuration**. In the left pane, select **Enterprise SSO**, and in the right pane, select **Enable Enterprise Single Sign-On**. All the settings will be unavailable. Select **Apply Configuration**, and then click **Next** to continue with the configuration.

Note
Services dependent on ENTSSO may be stopped after the upgrade. Check your System and Application event log for errors a nd warnings. You might have to restart any services that are dependent on the ENTSSO service on the computer where the u pgrade was performed.

Using Host-Initiated SSO Functionality in Enterprise Single Sign-On

Host-Initiated Single Sign-On uses the protocol transition feature of Windows Server 2003 to perform Single Sign-On for the non-Windows user. This feature requires Windows Server 2003 and must be in a domain that has its Domain Functional Level set to Windows Server 2003.

Supported Host Platforms
SNA connectivity, 3270 terminal emulation, host print service, and SNA programming interfaces

- IBM z/OS, OS/390, MVS, VSE and VM

SNA connectivity, 5250 terminal emulation, host print service, and SNA programming interfaces

- IBM i5/OS V5R4 & V6R1 and later

IP-DLC Link Service

- IBM z/OS 1.8 and later

OLE DB Provider for DB2, ODBC Driver for DB2, Managed Provider for DB2, and BizTalk Adapter for DB2

- IBM DB2 V8 and V9 for z/OS to support an SNA LU6.2 or TCP/IP network connection
- IBM i5/OS V5R4 & V6R1 to support an SNA LU6.2 or TCP/IP network connection
- IBM DB2 for Universal Database V8.2 and later for Windows and AIX operating systems to support a TCP/IP network connection

OLE DB Provider for AS/400 and VSAM, Host File Transfer ActiveX Control, Managed Provider for Host Files, and BizTalk Adapter for Host Files

- IBM Distributed File Manager (DFM) V1R4, V1R5, V1R6, and V1R7 to support an SNA LU6.2 network connection (DFM is a component of IBM Data Facility Storage Management Subsystem (DFSMS) for z/OS)
- IBM i5/OS V5R4 & V6R1 to support an SNA LU6.2 or TCP/IP network connection

AS/400 Data Queues ActiveX Control

- IBM i5/OS V5R4 & V6R1 to support an SNA LU6.2 network connection

Transaction Integrator and BizTalk Adapter for Host Applications

- IBM CICS Transaction Server for VSE/ESA V2R3 to support an SNA LU6.2 network connection
- IBM CICS Transaction Server for z/OS V2.3, V3.1, and V3.2 to support an SNA LU6.2 or TCP/IP network connection
- IBM IMS Version 9.1 & 10.1, with IMS Connect 2.2,to support an SNA LU6.2 or TCP/IP network connection
- IBM i5/OS V5R4 & V6R1 to support a TCP/IP network connection

WCF Channel for WebSphere MQ, MSMQ-MQSeries Bridge, and BizTalk Adapter for WebSphere MQ

- IBM WebSphere MQ for Windows V5.3 with Fix Pack 12
- IBM WebSphere MQ for Windows V6.0 with Fix Pack 6.0.1.1
- IBM WebSphere MQ for Windows v7.0

Deploying Host Integration Server

This section describes the concepts and considerations necessary for successfully deploying Host Integration Server.

In This Section

Deployment Overview

This section provides an overview of the SNA Open Gateway Architecture (SOGA) as it applies to Host Integration Server. It outlines the deployment models for Host Integration Server.

In This Section

SNA Open Gateway Architecture

Deploying Host Integration Server 2009

See Also

Other Resources

IP-DLC Link Service

SNA Open Gateway Architecture

Organizations with IBM mainframe or AS/400 computers are now consolidating their SNA-only and non-SNA networks into single TCP/IP-based wide area networks (WANs). In the past, two parallel networking systems have been deployed in most organizations: the traditional SNA network used to connect computers and hardware controllers with IBM mainframes and AS/400 computers, and local area networks (LANs) used primarily to access files and client/server applications running on local server computers.

Many organizations are seeking a single WAN solution that provides access to all desktops, servers, and host systems across a single networking protocol. At the same time, TCP/IP is emerging as the protocol of choice because of its versatility, openness, and ability to support Internet and intranet connections.

Many organizations cannot integrate their networks because of the need to ensure reliable and secure access to their existing IBM host systems—on which most organizations base their mission-critical daily operations.

SNA Open Gateway Architecture (SOGA) satisfies the requirement for reliable and secure host connectivity in the context of an integrated WAN. SOGA is a scalable framework for SNA enterprise connectivity, offering multiple options for integrating branch offices by means of routed inter-networks with IBM mainframe and AS/400 computers. This framework encompasses components from leading channel attachment and inter-networking vendors while deploying Host Integration Server computers in both traditional and innovative ways.

SOGA supports industry standards for SNA host access and presents flexible options for integrating host data across open WANs, such as those composed of Data Link Switching (DLSw), Frame Relay (RFC 1490), and Asynchronous Transfer Mode (ATM). Request for Comment (RFC) 1490 specifies methods of encapsulating TCP/IP and SNA within Frame Relay. The SOGA blueprint consists of three core elements:

- Flexible SNA gateway deployment models
- High-performance SNA gateway configurations
- Efficient wide area network utilization

See Also
Other Resources
Deploying Host Integration Server 2009

Deploying Host Integration Server 2009

Whether your enterprise consists of a single office or many branches spread throughout the world, SNA Open Gateway Architecture (SOGA) offers three flexible deployment models to implement Host Integration Server in your environment.

When planning your deployment strategy, ask the following questions:

- With what type of host systems do I need to connect?
- What kind of performance and host availability do my users expect?
- What kind of networking infrastructure is in place?

By answering these questions and using the information provided in this section, you can determine the best deployment strategy for your enterprise.

See Also
Concepts
SNA Open Gateway Architecture

Connecting Servers

After you have determined a suitable deployment model and subdomain configuration, you must choose how to connect Host Integration Server computers to the host system, to client workstations, and to other Host Integration Server computers.

This section explores how to choose:

- Connections between Host Integration Server computers and the host
- Network protocols for Host Integration Server computers and clients

To understand the connectivity options, you should first examine the different physical connections and network protocols that Host Integration Server supports.

In This Section

Connection Methods

Choosing Server-to-Host Connections

Choosing Network Protocols for Host Integration Server

Installing Host Integration Server 2009 Clients

See Also

Other Resources

IP-DLC Link Service

Connection Methods

Host Integration Server supports many different connection methods. However, not all may be available in your environment. When you choose a connection method, you should keep in mind:

- Deployment model you are implementing
- Host networking infrastructure that is in place
- Host systems to which you are connecting
- Expected usage level of host resources
- Level of performance and response expected by your users

In This Section

802.2 Data Link Control (DLC)

Synchronous Data Link Control (SDLC)

Channel

See Also

Other Resources

IP-DLC Link Service

802.2 Data Link Control (DLC)

Token Ring, Ethernet, and Fiber Distributed Data Interconnect (FDDI) connections use the IEEE DLC 802.2 protocol to connect to the host. In a mainframe environment, Host Integration Server usually connects to a front-end processor (FEP) or a communications controller, such as an IBM 3174. Occasionally, you can establish a connection to an adapter within a mainframe. With an AS/400 system, the connection goes directly to a network adapter in the AS/400 computer.

Token Ring connections are typically limited to 4 or 16 megabits per second (Mbps). Token Ring solutions are still found in some mainframe environments. In many scenarios, a Token Ring connection provides good performance at a reasonable cost in equipment.

Ethernet is a common connection type for local area networks (LANs) with workstations or computers. This connection type is also common in peer-type networks.

FDDI connections use fiber cabling to achieve connections reaching 100 Mbps or greater. The FDDI line communicates through an FEP that is channel-attached to the mainframe. This may limit the overall speed of communications. In many cases, this type of connection can provide a reasonably cost-effective solution if you require higher throughput and a channel solution is not possible.

DLC connections are well suited for the centralized and distributed deployment models, because both strategies implement Host Integration Server computers close to the host. You can also deploy data link control (DLC) in a branch model using routers or bridges that can route traffic between the branch offices and the central host system.

Host Integration Server supports DLC connections over any network adapter supported by Windows using the Windows-based DLC network transport driver.

See Also

Concepts

Synchronous Data Link Control (SDLC)

Channel

Other Resources

Connection Methods

Synchronous Data Link Control (SDLC)

SDLC connections use a standard phone line (leased, public, point-to-point, or multidrop). An SDLC adapter within the Host Integration Server computer connects to a modem that uses the phone line to establish a connection with the host system. In a mainframe system, the SDLC connection uses a front end processor (FEP), a communications controller, or an integrated SDLC adapter. In an AS/400 system, the connection goes directly into the AS/400 computer.

SDLC throughput is limited by the medium used for the connection and the capabilities of the SDLC adapter in the Host Integration Server computer. The cost to implement SDLC grows significantly as faster line types are used. In general, SDLC connections are much slower than 802.2 connections. The following table lists the common line types and their speeds:

Line type	**Typical speed**
Analog (conventional phone line)	9600 to 19200 bits per second (BPS)
Digital Data System (DDS)	56 to 64 kilobits per second (Kbps)
Integrated Services Digital Network (ISDN)	56 to 128 Kbps
T1 carrier system	1.544 megabits per second (Mbps)
T3 carrier system	2.048 Mbps

SDLC connections are useful for wide-area connections between geographically disparate locations or when bandwidth and usage requirements are low. Because of these factors, SDLC is ideally suited for branch-type deployment strategies.

Host Integration Server supports SDLC connections using the link support included with Host Integration Server or through an SDLC link service available through various third-party vendors. Not all supported SDLC adapters support all link speeds listed in the previous table.

See Also

Concepts

802.2 Data Link Control (DLC)

Channel

Other Resources

Connection Methods

Channel

Channel connections provide direct channel attachment to a mainframe. There are two types of channel connections: older Bus & Tagchannel connections, which can reach speeds of up to 4.5 megabytes per second (MBps), and newer ESCON connections, which can reach speeds approaching 17 MBps. Support for channel connections is limited to adapters supported natively through Host Integration Server.

Enterprise System Connection (ESCON) channel attachments use standard fiber cable and increase throughput significantly.

ESCON is an ideal connection method in centralized or distributed deployments where high performance and responsiveness are required.

The following table summarizes all common connection methods that can be used with Host Integration Server.

Method	Throughput	Characteristics
Token Ring	4 or 16 MBps	Midrange performance using data link control (DLC) protocol Supports multiple host connections using a single adapter Uncomplicated and inexpensive to implement Suitable for a wide range of purposes
Ethernet	10 GBps	Midrange performance using DLC protocol Supports multiple host connections using a single adapter Uncomplicated and inexpensive to implement Suitable for light to medium network traffic conditions
FDDI	100 MBps	High performance using DLC protocol Supports multiple host connections using a single adapter Relatively expensive to implement Suitable for higher-performance connections
SDLC	9600–19200 bits per second (BPS)	Low performance using SDLC protocol Supports 256 sessions over a single host connection Uncomplicated and inexpensive to implement Suitable for low-traffic WAN connections
Channel: Bus & Tag	4.5 MBps	High performance Supports a high number of host connections. More expensive Suitable for high-performance connections
Channel: ESCON	17 MBps	Highest performance Supports a high number of host connections More expensive Suitable for conditions where maximum throughput is required

See Also

Concepts

802.2 Data Link Control (DLC)

Synchronous Data Link Control (SDLC)

Other Resources

Choosing Server-to-Host Connections

In Host Integration Server terms, a host connection is the data communications path between Host Integration Server and a host system. For a mainframe, the connection corresponds to a physical unit (PU) definition in Virtual Telecommunications Access Method (VTAM). On the AS/400 computer, the connection corresponds to an Advanced Program-to-Program Communications (APPC) controller definition.

For each physical adapter or connection, you install and configure an appropriate link service within Host Integration Server. A link service is a Windows-based service or device driver that is used to control server-to-host communication adapters supported by Host Integration Server. Once configured, the link service is available for use not only on the configured Host Integration Server computer, but also on any server in the subdomain using the Distributed Link Service (DLC) feature. See Distributed Link Service for more information.

After you configure the link services, you can configure the connections. Using a host connection, a client computer on a LAN can communicate with the mainframe system. For some link services, it is possible to define multiple connections over a single host link.

In SNA terms, a physical unit (PU) is the combination of a physical connection and the link service it uses. In an SNA network, Host Integration Server provides PU 2 or PU 2.1 functionality similar to that of an IBM cluster controller.

Several factors should be taken into account when determining a physical connection method to your mainframe:

- Performance requirements
- Expected server loads
- Existing network infrastructure
- Chosen Host Integration Server deployment model
- Cost

You should plan for enough future capacity to support additional connections to your host system. Ethernet connections are the best choice for an all-purpose connection to a host.

For some link services, multiple host connections are possible using a single adapter, most notably the IP-DLC link service. Host Integration Server supports up to 250 host connections per server. Up to four instances of Host Integration Server are supported on a single computer.

In This Section

Mainframe Connection Summary

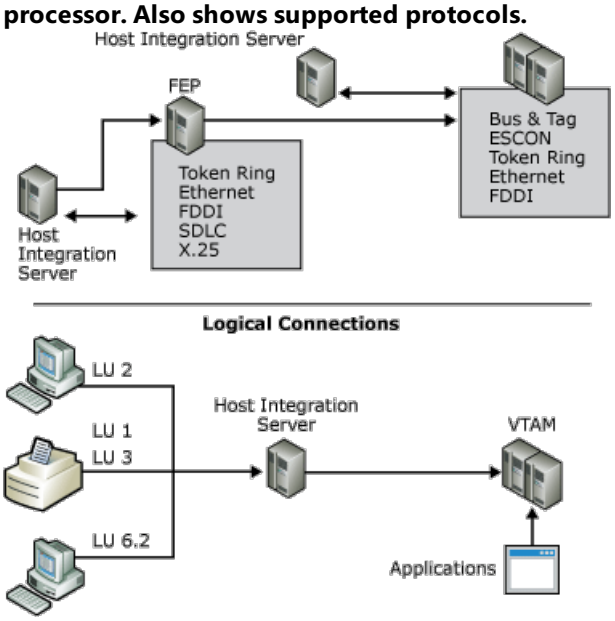
AS/400 Connection Summary

Mainframe Connection Summary

The hierarchical SNA network model provides access to a centralized processing resource from elements on the network. This model is most frequently associated with mainframe environments in which centralized applications are accessed from remote terminals across a network.

Devices in a hierarchical SNA network, such as terminals or cluster controllers, are called physical units (PUs). A number designates each class of device. For example, the mainframe is known as a PU 5 device.

Diagram of network model showing Host Integration Server connecting to a mainframe and to a front-end processor. Also shows supported protocols.



Host Integration Server can directly connect to the mainframe if a high-performance connection is required. These physical connection methods are available:

- Open Systems Adapter, supporting Token Ring, Ethernet, and Fiber Distributed Data Interconnect (FDDI) connections
- Bus & Tag channel connection
- Enterprise System Connection (ESCON) channel connection

Connections to a front-end processor (FEP), which is a PU 4 device, are also supported. These types of connections may be easier to implement depending on your existing infrastructure and the physical proximity of the Host Integration Server to the mainframe. For an FEP, you can use one of the following connection methods:

- Token Ring
- Ethernet
- FDDI
- SDLC

In a hierarchical SNA network, Host Integration Server emulates a cluster controller and supports all standard protocols:

- LU 2, for 3270 terminal sessions
- LU 1 or 3, for SCS or 3270 printer sessions
- LU 6.2, for Advanced Program-to-Program Communications (APPC) and Common Programming Interface for Communications (CPI-C) applications

- LU 0, 1, 2, or 3, for logical unit application (LUA) RUI/RI general-purpose, customized applications

Any combination of these protocols can be used over a given physical connection.

See Also

Concepts

AS/400 Connection Summary

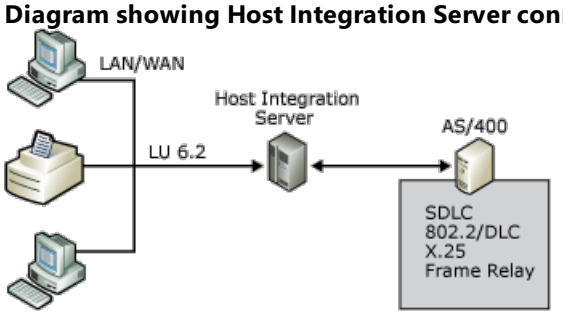
Other Resources

Choosing Server-to-Host Connections

AS/400 Connection Summary

In the peer-oriented SNA network model, all computers on the network can communicate directly with each other. Advanced Peer-to-Peer Networking (APPN) is the architecture developed by IBM that enables distributed data processing. APPN defines how components communicate with each other, and the level of network-related services, like routing sessions, that are supplied by each computer in the network.

Diagram showing Host Integration Server connecting to an AS/400 with several connection methods



In an APPN network, Host Integration Server emulates a type 2.1 physical unit device (PU 2.1). Host Integration Server computers can connect to an AS/400 computer using several connection methods:

- Token Ring
- Ethernet
- FDDI
- SDLC

Frame relay or bridging solutions can also be implemented to transport SNA traffic over wide area network (WAN) connections in branch-based deployment models. Host Integration Server operates as an APPN low-entry networking (LEN) node and communicates with other APPN nodes using the Advanced Program-to-Program Communications (APPC) or LU 6.2 protocol.

See Also

Concepts

Mainframe Connection Summary

Other Resources

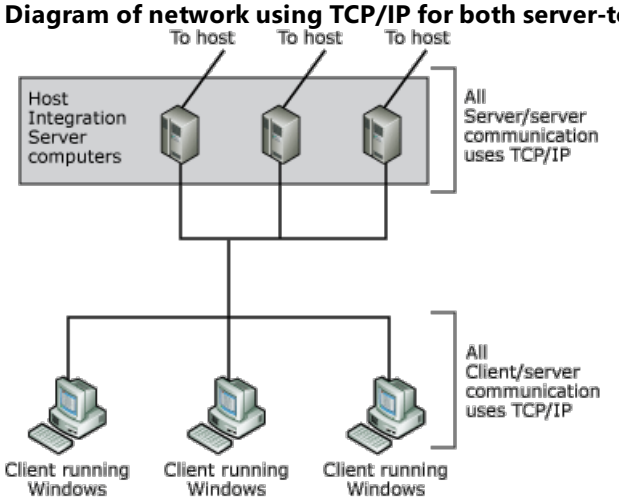
Choosing Server-to-Host Connections

Choosing Network Protocols for Host Integration Server

After you determine the server-to-host connection, you must choose the protocols to use for two additional Host Integration Server communication paths: client/server communication and server/server communication. You can use one protocol for both, or you can use a combination of different protocols, if all servers share at least one client/server protocol and use it for server/server communication.

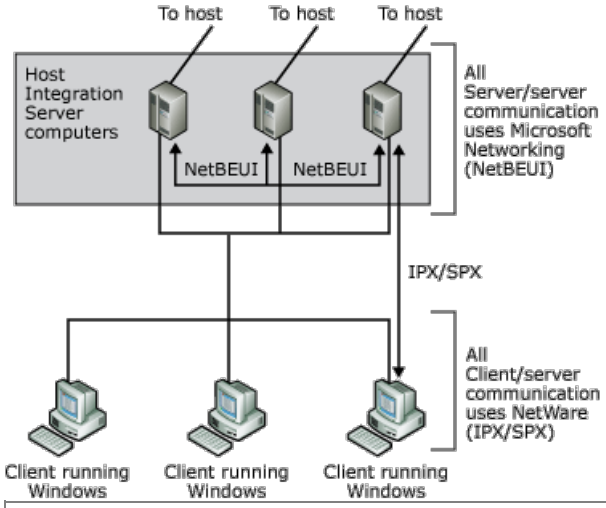
Deploying a single protocol across your wide area network (WAN) is the easiest way to manage your communications. The following figure shows a network in which TCP/IP is used for all types of communication involving Host Integration Server computers and clients.

Diagram of network using TCP/IP for both server-to-server and server-to-client communication



You may decide to gradually implement Host Integration Server throughout your enterprise, in which case you may need to use existing protocols for certain connections. The following figure shows a local area network (LAN) in which two protocols are used. For server/server communication, Microsoft Networking (Named Pipes) is used.

Diagram of network using NetBEUI for server-to-server communication and IPX/SPX for server-to-client communication



Note
IPX\SPX is no longer supported in this version of Host Integration Server.

Similarly, other combinations of protocols can be used, if all the Host Integration Server computers share one protocol and use that protocol for server/server and client/server communication.

In This Section

Choosing Client/Server Network Protocols

Choosing Server/Server Network Protocols

See Also

Other Resources

Choosing Client/Server Network Protocols

Host Integration Server client computers can communicate through a number of different local area APIs or network transports: Microsoft Networking (Named Pipes), or TCP/IP (sockets). The network software and the Host Integration Server software must be installed correctly on both clients and servers for them to handle different protocols correctly. Correct installation ensures two essential aspects of communication:

- Host Integration Server computers and client computers are visible to each other on the LAN. This results when the network software is installed correctly on all affected computers.

- Host Integration Server computers communicate with clients over the correct LAN protocol. Clients direct their communication to the correct subdomain name or (for some clients using Microsoft Networking (Named Pipes) or TCP/IP) to one or more correct server names.

TCP/IP is the standard network protocol for client/server applications. Its high performance and routing ability make it suitable for many wide area network (WAN) environments. In many cases, TCP/IP will be the best protocol choice for your network, especially if TCP/IP is already deployed to some degree in the LAN segment on which the Host Integration Server computers and clients reside.

See Also
Concepts
Choosing Server/Server Network Protocols
Installing Host Integration Server 2009 Clients
Other Resources
Choosing Network Protocols for Host Integration Server

Choosing Server/Server Network Protocols

Host Integration Server computers communicate with each other using mailslot messages or directed datagrams over a network. Host Integration Server enables you to make choices concerning these broadcasts.

You can select among the following protocols for server broadcasts: Microsoft Networking (Named Pipes), and TCP/IP.

Important
You must make sure that at least one protocol is available on all Host Integration Server computers in the subdomain, and th at this protocol is used for server/server communication and client/server communication.

Note
For example, if all Host Integration Server computers in a subdomain use TCP/IP, the protocol used for server/server commu nication must be TCP/IP, and all servers must use TCP/IP for client/server communication as well.

Using multiple server/server transport protocols can add significantly to network overhead. This is because every server broadcast must be sent out through all protocols selected in the Host Integration Server management console.

For highest efficiency, select only one protocol for broadcasts between Host Integration Server computers. Remember that the protocol must be available on all Host Integration Server computers in the subdomain. Where multiple choices are available (multiple protocols are bound to the network adapters on all Host Integration Server computers), select a protocol other than Microsoft Networking (Named Pipes). This is recommended because of design requirements for mailslot broadcasts. When these broadcasts are sent over Microsoft Networking (Named Pipes), there is the local system cannot determine which protocol the receiving system will be using, so the broadcast is sent over all protocols bound to the local adapter. This causes multiple mailslot broadcasts by means of Microsoft Networking (Named Pipes) for adapters to which several protocols are bound. The multiple broadcasts cause an increase in network traffic.

In subdomains in which all Host Integration Server adapters can use multiple protocols, TCP/IP protocol is recommended. You may be able to select extra protocols that do not exist in your network. However, selecting these has no effect.

The Host Integration Server management console also allows you to specify a parameter called Mean Time Between Server Broadcasts. For optimum efficiency, you should specify a value greater than or equal to 60 seconds (the default). The smaller this value, the less the efficiency, but the more likely the broadcasts will compensate for lost messages. It is recommended that you choose a value of 120 seconds or greater initially. If you encounter an increased number of lost messages, this value should be reduced until a low number of messages are lost.

See Also

Concepts

Choosing Client/Server Network Protocols

Installing Host Integration Server 2009 Clients

Other Resources

Choosing Network Protocols for Host Integration Server

Installing Host Integration Server 2009 Clients

Host Integration Server client software allows client workstations to communicate with Host Integration Server computers to access host resources. Client software is installed on each workstation using applications that communicate using any Host Integration Server programmatic interfaces. Client software is available for the following platforms:

- Microsoft Windows Server 2008
- Microsoft Windows Server 2003
- Windows 2000 Server

The fastest Host Integration Server client/server network interface is TCP/IP, although you can use other local area network (LAN) protocols such as Microsoft Networking (Named Pipes) if your LAN supports them. If you select TCP/IP or Microsoft Network, the remote installation option is recommended. Selecting "local" requires the client workstation to be on the same TCP/IP subnet as the Host Integration Server computer, which is uncommon in routed IP networks.

Note
Host Integration Server client software is not required to use services such as TN3270 and TN5250. Applications, such a TN3270 emulator, communicate directly with these services using TCP/IP and do not use the Host Integration Server client/server interface.

See Also
Concepts
Choosing Server/Server Network Protocols
Choosing Client/Server Network Protocols
Other Resources
Choosing Network Protocols for Host Integration Server

Understanding Connectivity

After you have completed Host Integration Server installation, you are ready to configure your installation. This section provides the concepts and procedures for configuring Host Integration Server.

In This Section

Host Integration Server 3270 Connectivity

Host Integration Server 5250 (AS/400) Connectivity

Independent APPC LUs

Dependent APPC LUs

APPC Mode Definition

CPI-C Access

Transaction Programs

APPC Security

Optimizing Communications

APPC Mode

Notes Section

See Also

Other Resources

IP-DLC Link Service

Host Integration Server 3270 Connectivity

In the hierarchical SNA network model most frequently associated with a mainframe computer, you access centralized applications from remote terminals across a network. This network model uses the information display protocol for IBM mainframe computers known as 3270. This protocol facilitates conversations between the mainframe and devices such as terminals, printers, and controllers. Through the definition and assignment of 3270 logical units (LUs), Host Integration Server provides access to mainframe resources.

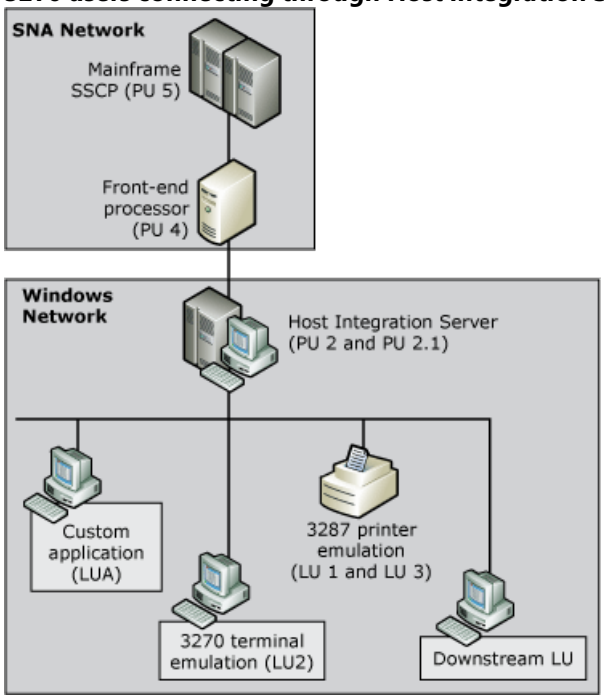
A 3270 LU is known as a dependent LU because it requires a mainframe to function. Each 3270 LU defined within Host Integration Server is configured to use an existing connection to the mainframe system. Each 3270 LU corresponds to a matching LU resource allocated on the host computer, usually specified within Virtual Telecommunications Access Method (VTAM). The 3270 LU definition in Host Integration Server is identified by a number that matches the number of the corresponding LU resource on the mainframe, and by a user specified name.

The 3270 LU is further classified by the type of service provided over the connection. Like physical units (PUs), numbers designate LU types. For example, 3270 display data streams are known as LU 2 streams. Within Host Integration Server, a 3270 LU can be configured as one of the following types:

- Display (LU 2)
- Printer (LU 1 or LU 3)
- Application (LUA)
- Downstream

After the LUs are configured, they are accessed from client computers and applications using Host Integration Server client software that is installed on the client workstation. The client software manages communications between a 3270 application (like a terminal emulator) and the Host Integration Server computer. Applications designed for the Host Integration Server client API use the LUs defined within Host Integration Server to establish a communications link from the client workstation to the mainframe by means of Host Integration Server.

3270 users connecting through Host Integration Server to a mainframe



The link between the LU definition in Host Integration Server and the host LU resource is called a session. Sessions can be permanent and automatically started during initialization, or established on an as-needed basis. Concurrent sessions can share the same physical devices and communications links.

A 3270 user communicating with a host uses a 3270 LU. The 3270 LU has a specific name (recognized by the host), is associated with a specific connection, and supports a specific use (either printing, or terminal emulation of a particular size). A collection of 3270 LUs that will be used by a group of users can be placed in an LU pool, so that whenever a user needs an LU,

the next one in the pool will be made available. The connection, the 3270 LU, the 3270 LU pool, and the list of users (or groups) are the basic elements to configure when supporting 3270 users in a Host Integration Server installation.

See Also

Concepts

LUA Access

Precedence of Accounts in Determining LU Access

Downstream Connections

LUA Access

Logical unit application (LUA) is an application programming interface (API) that allows you to write customized applications to communicate with the host. LUA LUs enable programmable control over the SNA messages that are sent between the communications software and the host. LUA LUs can be used to communicate with LU types 0, 1, 2, or 3 at the host as long as the application sends the appropriate SNA messages required by the host.

An LUA application uses a local LU, which uses Host Integration Server to communicate with the host system. When Host Integration Server connects to the host, there are three progressive sessions:

1. The PU-SSCP between the Host Integration Server physical unit (PU) and the system services control point (SSCP) on the host.

2. The SSCP-LU session between the LUA LU at the computer running the application and the SSCP.

3. The LU-LU session between the LUA LU at the computer running the application and the host LU.

The LU session's normal flow carries most of the data. The other flows are used only for control purposes.

The Host Integration Server configuration file contains information that is required for LUA applications to communicate. An LUA LU is configured to use a connection to the host by the link service that is installed. It is then given the LU number that matches that of an LU on the host.

The configuration may include LUA LU pools. A pool is a group of LUs with similar characteristics, and it allows an application to use any free LU from the pool. This feature can be used for allocating LUs on a first-come, first-served basis when there are more applications than LUs available, or for providing a choice of LUs on different connections, as well as providing hot backup and load balancing.

The communications components to be configured include:

- A link service for communicating with the host.

- A connection to the host that uses the link service.

- A Host Integration Server service which owns the connection. This component is configured automatically.

- An LUA LU on a Host Integration Server service, configured to use the connections, with an LU number that matches an LU on the host.

You can perform the following actions with LUA LUs:

- Assign an LUA LU, or a range of LUA LUs, to a connection. A range of LUA LUs will allow multiple applications to use LUAs simultaneously. Configure the numbering for a new range of LUA LUs by specifying the lowest LU number and the number of LUs in the range.

  After creating a range of LUs, you can change the numbering of individual LUs in the range.

- Configure properties for a new LUA LU or a new range of LUA LUs. This includes specifying the LU name and LU number for the LUs.

- Group LUA LUs into one or more LUA LU pools. An LUA LU pool contains a number of LUs that are made available as a group to LUA applications. A given application can get LU access as long as one of the pooled LUs is available.

- View or modify an existing LUA LU, regardless of whether it was created as part of a range.

- Copy properties from one LU to another.

- Move an LU from one connection to another.

- Delete an LU.
- Move an LUA LU from one connection to another.

Check with the host administrator to determine the appropriate names and numbers for LUA LUs on your system. The LU Number for LUs on 802.2 or Synchronous Data Link Control (SDLC) connections should match the LOCADDR= parameter of the LU definition in VTAM or in the NCP Gen.

If the number you specify has already been assigned to an LU or an APPC LU-LU pair on the current connection, you must use a different number. The range for LU numbers is from 1 through 254.

The LU Name cannot be the same as any other LU name or pool name (except for APPC LU names) on the server.

If the High Priority LU is assigned to a pool, the priority setting of the pool overwrites the setting of the LU.

On an 802.2 or SDLC connection, you can configure multiple LUs at one time by configuring them as a consecutively numbered range. Multiple LUA LUs will allow multiple applications to access the host simultaneously. After configuring the range of LUs, you can modify the numbering and properties of individual LUs in the range.

See Also

Concepts

Precedence of Accounts in Determining LU Access

Downstream Connections

Precedence of Accounts in Determining LU Access

When user and group account memberships overlap, the highest-priority account that contains a 3270 LU or pool determines the access the user gets. Accounts are prioritized as follows:

1. User accounts (highest priority)
2. Subdomain groups
3. Local groups
4. Well-known groups such as Everyone (lowest priority)

For example, if a 3270 LU called LU 1 is assigned to a user account (a high-priority account) called JOHND, and at the same time an LU called LU 2 is assigned to a local group (a low-priority account) of which JOHND is a member, JOHND will be given access to LU 1, not LU 2.

See Also
Concepts
LUA Access
Downstream Connections

Downstream Connections

A downstream connection enables a remote computer without a direct connection to the host computer to pass information back and forth using Host Integration Server as the gateway. To the downstream system, there appears to be a direct connection to the host. Host Integration Server accomplishes this by passing information back and forth between the downstream system and the host.

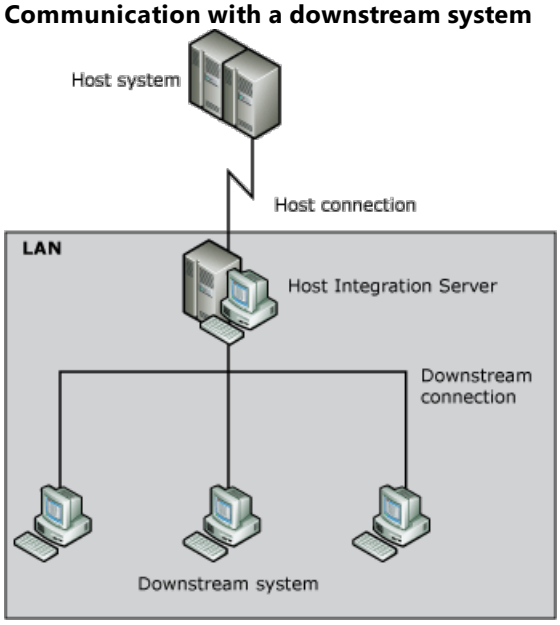
Downstream connections and LUs enable Host Integration Server to support communication between 3270 nodes using SNA protocols. A downstream system is an SNA node (a Host Integration Server computer or mainframe) that uses Host Integration Server as a physical unit (PU) gateway. To the downstream system, the Host Integration Server computer appears to be the 3270 host providing the PUs and LUs. The downstream system may be unable to communicate directly with the 3270 host because of hardware or network incompatibilities that are supported by the intermediate Host Integration Server computer.

One method of reducing host configuration requirements is to concentrate PUs on the SNA gateway computer and pass the LUs to attached downstream physical units (DSPUs). LUs from one or more PUs can be shared with one or more downstream devices. This allows for more economical use of configured resources and alleviates the need to configure each downstream device in host Virtual Storage Access Method (VSAM).

Communication by means of a downstream connection and downstream LU in Host Integration Server is always dependent (controlled by a host). Independent communication, including APPC, is not available by means of downstream LUs on Host Integration Server.

The following figure illustrates downstream connections.

Communication with a downstream system



The information that Host Integration Server passes from the downstream system to the host includes LU information. Therefore, Host Integration Server does not store detailed LU configuration information for downstream LUs. However, Host Integration Server does require all the usual connection information for the host and downstream connections.

Two connections are needed for a downstream system: a downstream connection (from the downstream system to Host Integration Server) and an upstream connection, which is an ordinary host connection from Host Integration Server to the host.

You can configure the upstream connection as you would any other connection to the host. After that, you can configure the downstream connections. These must be 802.2 or SDLC.

See Also

Concepts

LUA Access

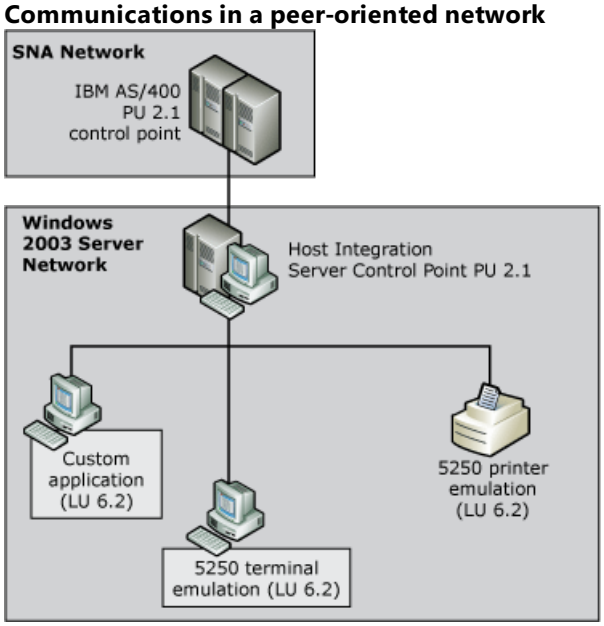
Precedence of Accounts in Determining LU Access

Host Integration Server 5250 (AS/400) Connectivity

In the peer-oriented SNA network model, all computers on the network can communicate directly with each other. Advanced Peer-to-Peer Networking (APPN) enables distributed data processing, defines how components communicate with each other, and determines the level of network-related services that are supplied by each computer in the network. Although peer-oriented SNA networks are usually associated with an AS/400 host system, mainframe systems can also support peer-to-peer networking.

IBM AS/400 computers use the 52xx series of devices. In particular, 5250 describes the terminal display data stream. The Advanced Program-to-Program Communications (APPC) protocol is used to support 5250 terminals and other APPN network computers, devices, and programs to communicate with each other. Each device in an APPN network is known as a type 2.1 physical unit (PU 2.1). In addition, the APPC protocol defines associated logical units as APPC LUs (also called LU 6.2).

Communications in a peer-oriented network



In an APPN network, Host Integration Server computers emulate PU 2.1 devices and can connect to an AS/400 using several connection methods:

- Token Ring
- Ethernet
- Fast Ethernet
- FDDI
- SDLC

Frame relay or bridging solutions can also be implemented to transport SNA traffic over wide area network (WAN) connections in branch-based deployment models.

See Also

Concepts

APPC

Connecting to an AS/400 Using 5250 Terminals

APPC

Advanced Program-to-Program Communications (APPC), or LU 6.2, provides a transport vehicle for programs on peer systems to communicate with each other over the SNA network. APPC is software that enables high-speed communications between programs on different computers.

APPC is a change from the terminal-to-host connections used in many 3270 systems. APPC moves to a distributed transaction programming environment by providing a common set of SNA protocols that brings compatibility at a program communications level.

APPC serves as translator between application programs and the network. When applications on one computer pass information to the APPC software, APPC translates the information and passes it to a network interface. APPC translates the information back into its original format and passes it to the corresponding partner application. APPC can be used across any of the standard types of connections supported by SNA and is not tied to any particular physical connection.

APPC generally uses a local APPC LU (LU 6.2) and one or more remote APPC LUs. A local APPC LU can be independent or dependent. In full peer-oriented APPN networks, typically implemented in an AS/400 environment or under the most modern evolution of mainframe technology, the independent APPC model applies. Dependent APPC is used in older mainframe networks, and its functions are reduced.

Local and remote APPC LUs work together in pairs. The local APPC LU is assigned to a server (unlike other LU types, which are assigned to connections). The remote APPC LU is assigned to the connection. Host Integration Server uses dynamic partnering to create any possible LU partnership on demand when local and remote LUs and modes recognize each other.

With dynamic APPC partnering, an administrator configures remote LUs, but does not need to partner them with local LUs. Host Integration Server will automatically partner the LUs when needed.

With dynamic APPC configuration, an administrator does not need to configure remote LUs. If a connection is designated to support dynamic APPC configuration, Host Integration Server will automatically define a remote LU and partner it with a local LU when needed.

Independent APPC provides the ability to run multiple, concurrent, parallel sessions between a single pair of LUs. Dependent APPC allows only a single session between a given pair of LUs.

Programs that use APPC are referred to as transaction programs (TPs). There are two types of TPs: those that can invoke a conversation, and those that can be invoked. A TP can provide any type of service: terminal emulation, data transfer, database query or update, and so on.

The characteristics that govern the interactions between TPs using an LU 6.2-LU 6.2 connection are determined by the mode associated with the connection. The mode can be associated in a fixed manner with a given LU, or it can be supplied by the invoking TP when a session is first initiated.

See Also
Concepts
Host Integration Server 5250 (AS/400) Connectivity
Connecting to an AS/400 Using 5250 Terminals

Connecting to an AS/400 Using 5250 Terminals

Host Integration Server computers provide access to AS/400 computers by emulating 5250 display terminals. A 5250 user communicating with an AS/400 must use a pair of APPC LUs (LU 6.2). This pair contains a local LU and a remote LU (as viewed by the Host Integration Server computer). Together, these two LUs (along with the mode that they use) contain the configuration information needed for establishing a session with the AS/400 computer.

A local APPC LU can be either independent or dependent. An independent local APPC LU can communicate directly with a peer system. A dependent APPC LU requires the support of a mainframe. These are described in more detail in later sections.

The 5250 user can share this pair of LUs with many other users at the same time, or the 5250 user can have exclusive possession of the LUs (depending on the Host Integration Server configuration). In addition, Host Integration Server computers can be configured so that users can start 5250 emulation sessions without knowing the names of LUs to request. This configuration is accomplished with the use of default LUs that are specified for each 5250 user or for groups of users.

See Also

Concepts

Host Integration Server 5250 (AS/400) Connectivity

APPC

Using Wizards

Using Wizards

You can configure 5250 terminal emulation using the wizard provided with Host Integration Server. Available under the **Tools** menu, the wizard takes you through each step of configuring AS/400 connection properties, creating remote APPC LUs linked to your AS/400 computer, and, if necessary, creating local APPC LUs.

In addition to wizards, several features of Host Integration Server can simplify configuration for APPC:

- Implicit Incoming Remote LU and Implicit Incoming Mode, which allow Host Integration Server to accept requests that arrive by unrecognized remote LUs and modes.

- Default Local APPC LU and the Default Remote APPC LU, which allow LU aliases to be associated with user or group names, simplifying the routing of incoming requests and the configuration of client systems.

- Default Outgoing Local APPC LU Pool, which allows LUs to be allocated dynamically to any invoking TP (transaction program) that does not specify a local LU.

- Single-System APPC, which allows two local APPC LUs residing on the same server to communicate with each other using defined parameters.

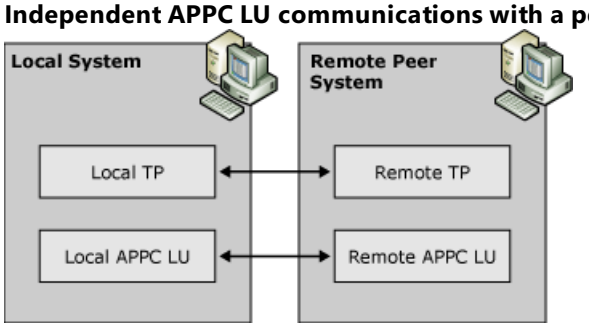
Independent APPC LUs

In APPC, an independent LU can communicate directly with a peer system and does not need the support of a host computer.

Independent APPC LUs provide the ability to run multiple, concurrent, parallel sessions between a single pair of LUs. Programs that use independent APPC LUs are referred to as transaction programs (TPs) and may provide any type of service: terminal emulation, data transfer, database query or update, and so on.

The following figure illustrates how an independent local APPC LU can communicate directly with a peer system.

Independent APPC LU communications with a peer system



Modes determine the interactions between TPs on an APPC-APPC connection. Modes can be assigned to an LU or supplied when the session is first established. For more information about modes, see APPC Mode Definition later in this section.

When configuring independent APPC LUs, note the following:

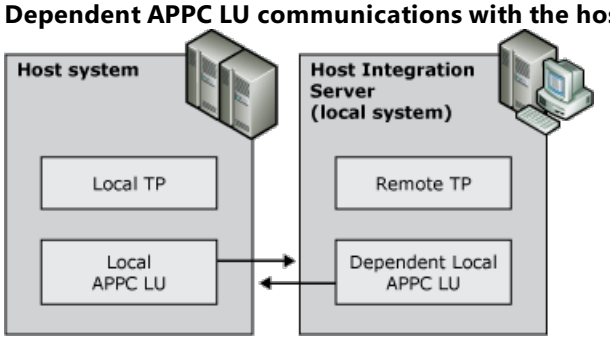
- If the independent local APPC LU communicates with a TP on a mainframe, the host system must be using VTAM version 3, release 2 or later with either NCP version 5, release 2 or later (3745 systems), or NCP version 4, release 3 or later (3725 systems).
- For independent LUs that communicate with a host, the LOCADDR parameter should be set to 0 in VTAM or in the NCP Gen.
- On connections used for independent APPC, the exchange identification (XID) type must be Format 3.
- The Network Name parameter is required. The default is the network name of Host Integration Server.
- An LU Name parameter is required to identify local and remote APPC LUs to other components on the network. In contrast to 3270 LUs, independent APPC LUs do not need an LU number.
- If the remote APPC LU supports parallel sessions, it can only be used with a mode whose parallel session limit has a value greater than 1.

Dependent APPC LUs

A dependent local APPC LU requires the support of a mainframe to communicate with a remote TP. Dependent APPC LUs cannot be used to communicate with an AS/400 computer.

The next figure illustrates how a dependent APPC LU can communicate with the host.

Dependent APPC LU communications with the host



Dependent local LUs have the following characteristics:

- Allow only a single session per LU.
- Use a connection configured with a Remote End of Host, not Peer.
- Only type of APPC LUs that Host Integration Server supports when communicating with a mainframe using a version of VTAM earlier than version 3, release 2.
- Require the mainframe VTAM to use a value of 1 or greater in the LOCADDR parameter in the NCP Gen.

The Network Name and LU Name are recommended but not required, because they are used only by local software such as the Microsoft® Windows® event log software. The default is the network name specified for the Host Integration Server computer. If a network name is not specified for the server, the default is APPN.

For a remote APPC LU that will be partnered with a dependent local APPC LU, the LU Name is recommended but not required. It identifies the LU to local software, such as the Windows event log software.

A number identifying the LU on its connection is required and should match the LOCADDR=parameter for the LU definition in VTAM or in the NCP Gen. Check the numbering for any non-APPC LUs on the connection that the remote APPC LU will use because this number must be unique.

The usual range for LU numbers is from 2 through 254.

A dependent APPC LU requires the support of a mainframe. During LU configuration, you need to select a host type of the connection to which the LU will be assigned. Also, the remote LU needs the same name as defined on the remote system services control point (SSCP). This name is required when using dependent APPC.

See Also

Concepts

Configuring Dependent LUs

Configuring Dependent LUs

When you configure a dependent LU, make sure you do the following:

- Set up a single session for each dependent LU. (Only one session is allowed for each dependent LU.)
- Specify the remote end, using **Host**.
- Use VTAM 3.2 or later for host-to-APPC LU communication.

Set the host VTAM to a value of 1 or greater in the *LOCADDR*= parameter of the LU definition.

SeeAlso

Dependent APPC LUs

APPC Mode Definition

A mode is associated with an LU-LU pair, and determines the session properties for that pair. One of the key mode properties is the Parallel Session Limit. This limit determines whether an LU-LU pair can perform only one interaction at a time or multiple concurrent interactions.

Parallel sessions are used only with independent APPC. If parallel sessions are to be allowed with an LU-LU pair, the local LU must be independent, and the remote LU in the pair must support parallel sessions.

If the LU-LU pair can have multiple parallel sessions, other mode properties, such as Minimum Contention-Winner Limit, determine to what extent each LU can initiate interactions.

The following table lists the modes provided with Host Integration Server.

Mode name	To be used for
#BATCH	Batch-oriented sessions
#BATCHC	Batch-oriented sessions using compression
#BATCHSC	Batch-oriented sessions that employ a minimal level of routing security
BLANK	Sessions using a default mode name, encoded as eight blank EBCDIC spaces in BIND
#INTER	Interactive sessions
#INTERC	Interactive sessions using compression
#INTERSC	Interactive sessions that employ a minimal level of routing security
QPCSUPP	All sessions with an AS/400 computer
QSERVER	The ODBC drivers

The configuration parameters of the modes provided with Host Integration Server are shown in the following table.

Configuration parameter	#BATCH #BATCHSC and BLANK	#INTER and #INTERSC	QPCSUPP	QSERVER
Parallel Session Limit	8	8	64	8
Minimum Contention Winner Limit	4	4	32	4
Partner Contention Winner Limit	0	0	0	0
Automatic Activation Limit	0	0	0	0
High Priority Mode	No	Yes	Yes	Yes
Pacing Send Count	3	7	7	7
Pacing Receive Count	3	7	7	7
Max Send RU Size	1024	1024	1024	1024
Max Receive RU Size	1024	1024	1024	1024

The mode name and configuration parameters are used for both dependent and independent APPC LUs.

The Parallel Session Limit defines the number of sessions that can be activated. If the local APPC LU is dependent, specify 1 for the parallel session limit. Dependent local APPC LUs cannot have parallel sessions. Independent local APPC LUs can have from 1 through 1024 parallel sessions. The default is 1.

The Minimum Contention Winner Limit is the guaranteed number of sessions the local machine can initiate. The Partner Contention Winner Limit is the guaranteed number of sessions the remote machine can initiate. Each session can be established without permission from the partner LU. The sum of both must be less than or equal to the Parallel Session Limit. The range for each is from 0 through the Parallel Session Limit; the default is 0.

The Automatic Activation Limit specifies the number of Contention Winner sessions to be activated for the local LU whenever the connection for this mode is started. In a Contention Winner session, the LU can initiate conversations without permission from the partner LU. For a single-system APPC (communication between two local LUs), this field is meaningless. The range is from 0 through the Minimum Contention Winner limit.

High Priority Mode defines the priority of the data sent. This is beneficial for segregating batch data from interactive data. Batch data typically has a lower priority than interactive data.

Pacing Send Count and Pacing Receive Count allow you to specify the maximum number of frames without an SNA pacing response from either the local or remote APPC LU. For example, if you specify 0, the local APPC LU can send an unlimited number of frames without receiving a response. In this case, the remote APPC LU can negotiate and set a limit on the count. The range is from 0 through 63; the default is 4.

The request/response unit (RU) sets the size of the data message that can be sent or received. If an application needs to send a file that is larger than the specified size, it needs to break it up before sending the file. The minimum RU size on Host Integration Server is 256. The range for Max Send RU Size and Max Receive RU Size is from 256 through 16384; the default is 1024.

To add an APPC mode definition

1. In the SNA Manager tree, click **APPC Modes**.
2. On the **Action** menu, point to **New**, and click **Mode Definition**.
3. On each tab, configure the mode properties.
4. On the **Action** menu, click **Save configuration**.

See Also
Tasks
Single-System APPC
Concepts
Implicit Incoming Remote LU and Implicit Incoming Mode
Default Local APPC LU and the Default Remote APPC LU
Default Outgoing Local APPC LU Pool

Implicit Incoming Remote LU and Implicit Incoming Mode

An implicit incoming remote LU is a remote APPC LU that defines the properties to use when Host Integration Server receives a request to start a session with a local LU and when the remote LU named in the request is not recognized by Host Integration Server.

An implicit incoming mode is a mode that defines the properties to use when Host Integration Server receives a request to start a session and the mode named in the request is not recognized by Host Integration Server.

An implicit incoming remote LU requires an implicit incoming mode. An implicit incoming mode must be configured for any remote LU that will be used as an implicit incoming remote LU. An implicit incoming mode can be (but does not have to be) configured for remote LUs that will only be used explicitly.

You may want to accept incoming requests that arrive by many different remote LUs without having to explicitly define each remote LU on a Host Integration Server computer. The Implicit Incoming Remote LU and Implicit Incoming Mode allow for flexible acceptance of requests. When these two items have been configured, and an incoming request is received by Host Integration Server, the remote LU need not be recognized, as long as the local LU specified in the request is recognized. In such a situation, Host Integration Server uses the properties of the Implicit Incoming Remote LU and Implicit Incoming Mode for that LU-LU session.

For a session to be established, the incoming local LU name must be recognized by Host Integration Server. Then the incoming remote LU must either be recognized explicitly or handled implicitly (if an implicit incoming remote LU has been configured). If the remote LU is recognized explicitly, but the mode is not recognized (as part of an LU-LU pair), Host Integration Server internally creates a new mode definition with the correct name using the properties of the implicit incoming mode. Alternatively, if the remote LU is handled implicitly, Host Integration Server also handles the mode implicitly, by internally creating a mode, as described.

To use an implicit incoming remote LU

1. In the SNA Manager tree, expand an SNA subdomain, expand the server you want to work with, expand **SNA service**, and then expand **Local APPC LUs**.
2. Right-click the first local LU that you want to configure, and click **Properties**.
3. Click the **Advanced** tab, and specify the **Implicit Incoming Remote LU** for incoming conversations, and then click **OK** to exit.
4. Using steps 2–4, configure each local LU that the remote system will call.
5. Expand **Remote APPC LUs**, right-click the remote LU that you want to configure, and then click **Properties**.
6. Click the **Options** tab, and select the **Implicit Incoming Mode**.
7. On the **Action** menu, click **Save configuration**.

Note
All incoming requests must specify a local LU name that is recognized by Host Integration Server, even when using an implici t incoming remote LU and implicit incoming mode.

See Also

Tasks

Single-System APPC

Concepts

APPC Mode Definition

Default Local APPC LU and the Default Remote APPC LU

Default Outgoing Local APPC LU Pool

Default Local APPC LU and the Default Remote APPC LU

If a user or group uses transaction programs, 5250 emulators, or APPC applications, you can assign a default local APPC LU and a default remote APPC LU. These default LUs are accessed when the user or group member starts an APPC program (a TP, 5250 emulator, or APPC application) and the program does not specify LU aliases.

With Host Integration Server, you can assign a default local APPC LU and a default remote APPC LU to each user or group. A user or group member can then start APPC programs (TPs, 5250 emulators, or APPC applications) that do not specify LU aliases. Host Integration Server will use the default local APPC LU and the default remote APPC LU assigned to that user or group.

There are two steps before assigning default APPC LUs to a user or group:

1. The user or group must have an account on the local area network.

2. The user or group must be added to the list used by Host Integration Server.

If a user is assigned LUs through one or more accounts, such as group accounts and the user's individual account, one account determines the access for that user. The account that determines this access is the account found first when searching is performed in this order:

- User accounts (highest priority)
- Domain groups
- Local groups
- Well-known groups such as Everyone (lowest priority)

For example, suppose a user account (a high-priority account) called JOHND contains LOCLU1 as the default local APPC LU, but no default remote APPC LU. At the same time, suppose a local group (a low-priority account) of which JOHND is a member contains LOCLU2 as the default local APPC LU, and REMLU2 as the default remote APPC LU. For JOHND, the high-priority assignment, the default local APPC LU of LOCLU1 will be combined with the only other available assignment, the default remote APPC LU of REMLU2.

See Also

Tasks

Single-System APPC

Concepts

APPC Mode Definition

Implicit Incoming Remote LU and Implicit Incoming Mode

Default Outgoing Local APPC LU Pool

Default Outgoing Local APPC LU Pool

With Host Integration Server, you can set up a pool of local APPC LUs that will be allocated dynamically for any invoking transaction program that does not specify a local LU. This feature is designed to coexist with the default local and remote LUs for users and groups, and does not override these settings.

Both of the default APPC LU features (the default outgoing local APPC LU pool and the default local and remote LUs for users and groups) allow client computers to start a session without specifying an LU name. This helps to simplify administration by eliminating APPC configuration for client computers.

If an application begins the invoking process but does not specify a local APPC LU, Host Integration Server first tries to determine the default local APPC LU of the associated user or group (the user or group logged on at the system where the application is running). If no such LU can be substituted, Host Integration Server attempts to assign an LU from the default pool of outgoing local APPC LUs. If no LUs are available from the default pool, the attempt to begin the invoking process is rejected.

The default outgoing local APPC LU pool differs from, and should not be confused with, 3270, LUA, and downstream pools.

See Also

Tasks

Single-System APPC

Concepts

APPC Mode Definition

Implicit Incoming Remote LU and Implicit Incoming Mode

Default Local APPC LU and the Default Remote APPC LU

Single-System APPC

Single-system APPC has two local APPC LUs residing on the same server. The LUs communicate with each other using defined parameters. You do not need to define a connection because the LUs are communicating with each other on the same system.

For a single-system APPC, you will need to assign two local APPC LUs to a server, not to a connection. Configuration of the local APPC LUs includes specifying the LU Alias, Network Name, and LU Name. Each of these labels identifies an LU to a particular set of software components.

If Common Programming Interface for Communications (CPI-C) is used, configure one or more CPI-C symbolic destination names. For more information, see CPI-C Access in the next section.

Single-system APPC is generally used for testing transaction programs. The basic steps for configuring single-system APPC are as follows.

To configure single-system APPC

1. Assign two local APPC LUs to a server.
2. Configure the local APPC LUs by specifying the LU Alias, Network Name, and LU Name.
3. If an appropriate mode has not already been configured, configure it.

Note
For single-system APPC, the Automatic Activation Limit specified in the mode is meaningless. Use one of the local LUs for the originator conversation and the other local LU for the destination conversation.

See Also

Concepts

APPC Mode Definition
Implicit Incoming Remote LU and Implicit Incoming Mode
Default Local APPC LU and the Default Remote APPC LU
Default Outgoing Local APPC LU Pool

CPI-C Access

Common Programming Interface for Communications (CPI-C) provides a consistent application programming interface for network applications. CPI-C is a set of C-language routines that applications distributed across an SNA network can use to communicate as peers and exchange data to accomplish a processing task.

CPI-C programming provides a mechanism, called side information, that associates a set of parameters with a specified symbolic destination name. The CPI-C program then uses the symbolic destination name to initialize a conversation.

If you are using applications based on CPI-C, you configure one or more CPI-C symbolic destination names. A CPI-C symbolic destination name is a name assigned to a set of properties for a CPI-C conversation.

LU partners will need to recognize each other. This is accomplished using fully qualified names and LU aliases. The fully qualified name is a two-part name that identifies the remote LU. The first part of the fully qualified name is the network name and the second part is the remote LU name.

You will also need a mode name. For more information about modes, see the section APPC Mode Definition earlier in this section.

Conversation-level security requires a match between the user ID and password supplied by the invoking TP, and the user ID and password stored on the server where the remote TP resides. If the ID and password do not match, the session is not activated.

Transaction Programs

The part of an application that initiates or responds to APPC communications is called a transaction program (TP). TPs use APPC to exchange data with other TPs on a peer-to-peer basis.

Similar to a conversation when people talk with each other, the communication between two transaction programs is called a conversation. An application running on your computer can have many conversations active at one time, either with one other transaction program or with different transaction programs.

There are two types of TPs: TPs that can invoke (initiate a conversation with) other TPs, and TPs that can be invoked. A TP that can invoke another TP is called an invoking TP, and a TP that can be invoked is called an invokable TP.

If your Host Integration Server installation contains multiple systems (client computers or Host Integration Server computers), you can place invokable TPs on more than one system. When an invoking request is received in such an installation, there will (potentially) be a choice of systems on which to run the invokable TP. You can maintain specific control over this choice, or you can allow the choice to be made randomly by Host Integration Server (to distribute the load).

You can maintain specific control over this choice of system by setting up invokable TPs with unique names, or by setting up each invokable TP to run only with a specific, unique LU alias. With this arrangement, the information provided by the invoking TP (in the ALLOCATE verb) can specify the particular system on which the TP should run.

You can avoid controlling this choice of system, and allow the choice to be made randomly by Host Integration Server, by setting the **DloadMatchLocalFirst** registry entry to **NO**, as described in the *Host Integration Server Administrators Reference*. Then use invokable TPs that leave the local LU alias unspecified. When an incoming request is received, it is routed randomly, rather than preferentially, to the local Host Integration Server computer. In addition, no matter what LU alias is requested for the invokable TP, there cannot be a mismatch. Host Integration Server will start the TP, choosing randomly among the available systems.

Following are three of the possible ways that TPs can be arranged to run.

In This Section

TP Name Unique for Each TP

If you want to specify the system on which the invokable transaction program (TP) will run, you can use a unique TP name for each invokable TP. In this arrangement, the invoking TP identifies the invokable TP (and system) by naming the TP. This makes it unnecessary for an invokable TP to specify any LU alias in registry or environment variables.

See Also

Concepts

TP Name Not Unique; Local LU Alias Unique
TP Name Not Unique; Local LU Alias Unspecified
Invoking Transaction Programs
Invoking TPs and Host Integration Server Configuration
Invokable Transaction Programs
Invokable TPs and the Host Integration Server Configuration

Other Resources

Transaction Programs

TP Name Not Unique; Local LU Alias Unique

Instead of using a unique transaction program (TP) name to specify the system on which the invokable TP will run, you can give the same name to multiple invokable TPs, but associate each TP with a unique local LU alias. Configure each invokable TP (through registry or environment variables) to use a unique local LU alias. Then set up the invoking TPs so that each one is routed not only to the correct TP name but also to the correct partner LU alias for the intended invokable TP.

See Also

Concepts

TP Name Unique for Each TP
TP Name Not Unique; Local LU Alias Unspecified
Invoking Transaction Programs
Invoking TPs and Host Integration Server Configuration
Invokable Transaction Programs
Invokable TPs and the Host Integration Server Configuration

Other Resources

Transaction Programs

TP Name Not Unique; Local LU Alias Unspecified

If it does not matter on which system an invokable TP runs, use the same name for multiple invokable TPs, but do not specify an LU alias in the registry or environment variables for the TPs. In such a situation, there are no associated aliases in the list of available invokable TP names on a Host Integration Server computer. Thus, a request received from an invoking TP cannot cause a mismatch on the LU alias, and will match according to the TP name.

If you set the **DloadMatchLocalFirst** registry entry to **NO**, as described in the *Microsoft Host Integration Server Reference*, the server randomly routes the request to one of the available TPs. This spreads the processing load among multiple systems, and provides hot backup (so that you can take systems online and offline without disrupting service).

See Also

Concepts

TP Name Unique for Each TP

TP Name Not Unique; Local LU Alias Unique

Invoking Transaction Programs

Invoking TPs and Host Integration Server Configuration

Invokable Transaction Programs

Invokable TPs and the Host Integration Server Configuration

Other Resources

Transaction Programs

Invoking Transaction Programs

An invoking TP initiates a conversation with other TPs. An invoking TP can be located on any system on the SNA network.

An invoking TP identifies itself by issuing a TP_STARTED verb. TP_STARTED specifies the name of the invoking TP, and may specify the LU alias that the TP uses (or may leave the LU alias blank).

Next, the invoking TP initiates the invoking process by issuing an ALLOCATE verb. In ALLOCATE, the invoking TP specifies the name of the invokable TP, and may also specify the partner LU alias (the LU alias to be used by the invokable TP).

See Also

Concepts

TP Name Unique for Each TP
TP Name Not Unique; Local LU Alias Unique
TP Name Not Unique; Local LU Alias Unspecified
Invoking TPs and Host Integration Server Configuration
Invokable Transaction Programs
Invokable TPs and the Host Integration Server Configuration

Other Resources

Transaction Programs

Invoking TPs and Host Integration Server Configuration

For Host Integration Server to support the beginning of the invoking process (to accept the TP_STARTED and ALLOCATE verbs issued by an invoking TP), the following parameters must be configured correctly:

- If the invoking TP specifies the LU alias that it uses (in TP_STARTED), that LU alias must match a local APPC LU alias on the supporting Host Integration Server computer.

- If the invoking TP leaves the LU alias blank in TP_STARTED, one of two methods for designating a default LU must be used on the supporting Host Integration Server:

    - Assign a default local APPC LU to the user or group that starts the invoking TP (the user or group logged on at the system from which TP_STARTED is issued).

      – or –

    - Designate one or more LUs as members of the default outgoing local APPC LU pool.

  If the invoking TP leaves the LU alias blank in TP_STARTED, Host Integration Server first attempts to determine the default local APPC LU of the associated user or group, and then attempts to assign an available LU from the default outgoing local APPC LU pool. If these attempts fail, Host Integration Server rejects the TP_STARTED request.

- In most situations, the supporting Host Integration Server must contain an appropriate connection to another system (host or peer). Sometimes, for testing purposes, Host Integration Server contains two local LUs paired together (for invoking and invokable TPs that are in the same domain). In this situation, a connection to a host or peer is not necessary.

- If the invoking TP specifies the partner LU alias (in ALLOCATE), that LU alias must match a remote LU alias. In addition, that remote LU alias must be paired with the local LU alias specified in TP_STARTED. If the LU alias is left blank in ALLOCATE, a default remote APPC LU must be assigned to the user who started the invoking TP. If the default remote APPC LU is used, it must be paired with the local LU that will be used. Otherwise, the ALLOCATE verb fails.

The preceding parameters support the beginning of the invoking process. For the invoking process to successfully complete, additional parameters must be configured on another Host Integration Server computer, as described in the next section.

See Also
Concepts
TP Name Unique for Each TP
TP Name Not Unique; Local LU Alias Unique
TP Name Not Unique; Local LU Alias Unspecified
Invoking Transaction Programs
Invokable Transaction Programs
Invokable TPs and the Host Integration Server Configuration
Other Resources
Transaction Programs

Invokable Transaction Programs

An invokable TP is a TP that can be invoked by another TP. Invokable TPs are written or configured to supply their names to Host Integration Server as a notification that they are available for incoming requests. Host Integration Server invokable TPs can be run on any Host Integration Server computer or on a Windows Server™ 2003 or Windows 2000 client.

There are two types of invokable TPs:

- **Operator-started invokable TPs**

  An operator-started invokable TP must be started by an operator before the TP can be invoked. When the operator-started invokable TP is started, it notifies Host Integration Server of its availability by issuing a RECEIVE_ALLOCATE verb. RECEIVE_ALLOCATE provides the name of the invokable TP.

- **Autostarted invokable TPs**

  An autostarted invokable TP can be started by Host Integration Server when needed. The TP must be registered on its local system, so that it can be identified to Host Integration Server. (For details about how the TP is registered, see *Microsoft Host Integration Server APPC Applications* or *Microsoft Host Integration Server CPI-C Applications*.) The registered information defines the TP as autostarted, and must specify the TP name. The registered information may also specify the local LU alias that the invokable TP will use.

  If no local LU alias is registered with autostarted TPs, the resulting Host Integration Server configuration can be more flexible in responding to invoking requests.

  After an autostarted invokable TP is started by Host Integration Server, the TP issues RECEIVE_ALLOCATE (just as an operator-started TP does). RECEIVE_ALLOCATE must provide the same TP name as was registered for the TP.

Each Host Integration Server maintains a list of invokable TP names and any LU aliases associated with the TP names. When a request comes in from an invoking TP, Host Integration Server compares the requested invokable TP name and the associated LU alias to the list of available invokable TPs (which may include associated LU aliases). For details about how this comparison is carried out, see *Microsoft Host Integration Server APPC Applications* or *Microsoft Host Integration Server CPI-C Applications*.

If a match is found, Host Integration Server signals the system containing the requested TP to connect to that Host Integration Server computer.

If no match is found, Host Integration Server rejects the incoming request.

See Also

Concepts

TP Name Unique for Each TP
TP Name Not Unique; Local LU Alias Unique
TP Name Not Unique; Local LU Alias Unspecified
Invoking Transaction Programs
Invoking TPs and Host Integration Server Configuration
Invokable TPs and the Host Integration Server Configuration

Other Resources

Transaction Programs

Invokable TPs and the Host Integration Server Configuration

For Host Integration Server to receive requests from an invoking TP on another system, and then route those requests to an invokable TP, certain parameters must be configured correctly:

- Host Integration Server must have a connection to the system from which the invoking TP's request is sent.

- Host Integration Server must have a remote LU capable of receiving the incoming request. This remote LU can be configured either explicitly or implicitly. When configured explicitly, there is an explicit match between a remote LU alias on Host Integration Server and the alias of the LU that conveys the invoking TP's request.

  When configured implicitly, an Implicit Incoming Remote LU (with its Implicit Incoming Mode) is used. Several items must work together. First, the LU alias specified in the incoming request (the LU alias requested for the invokable TP) must match a local LU alias on the server receiving the request. Second, the local LU on the server must have an Implicit Incoming Remote LU assigned to it. The properties of the Implicit Incoming Remote LU will be used for that LU-LU session. For more details about how an Implicit Incoming Remote LU works, see Implicit Incoming Remote LU and Implicit Incoming Mode, earlier in this section.

- Appropriate local LUs must be defined in the Host Integration Server configuration file.

APPC Security

Owners of APPC transaction programs may want to allow only a limited set of users to start the program. APPC provides a mechanism, called APPC conversation security, by which the client transaction program identifies its user to the server system.

There are three levels of security for client programs: None, Same, and Program:

- If the level of security is None, the client system sends no security information (user ID or password).
- If the level of security is set to Same, APPC tries to determine a user ID for the client program. If the server system requires a password and the client system permits APPC to retrieve one for the user ID, APPC will also send the password to the server system. If no user ID is available, or if the server requires a password but the client system does not allow APPC to retrieve the password, no security information is sent to the server. This is sometimes referred to as downgraded to security NONE.
- If the security level is set to Program, the client transaction program will override any security information that the local system may provide. The client program must supply both a user ID and a password. CPI-C programs may get the user ID and password by either prompting the user to enter the information or by checking the CPI-C side information. Not all systems allow this option.

If the client program uses SECURITY=SAME or SECURITY=PROGRAM, APPC on the server must check the user ID and password regardless of the server transaction's security requirements. This requirement can cause unexpected problems and is not recommended.

To configure session security for remote LUs

1. In the SNA Manager tree, expand an SNA subdomain, expand the server you want to work with, expand **SNA service**, and then expand **Remote APPC LUs**.
2. Right-click the LU that you want to configure, and click **Properties**.
3. Click the **Options** tab. Under **Session-Level Security**, select an option.
4. Click **OK**.
5. On the **Action** menu, click **Save configuration**.
6. To put the changes into effect, you must restart the server.

Note
Setting the security for remote LU sessions is optional.

See Also
Other Resources
Transaction Programs

Optimizing Communications

Servers used primarily for communications need to provide fast throughput, but they do not need to provide fast file access (as a file server would). Faster throughput will result if portions of memory are set aside for communications programs (such as Host Integration Server or Microsoft® SQL Server™).

Such dedicated memory includes nonpaged memory, or portions of memory that are never swapped to disk, but remain available for immediate use at all times. If more memory is dedicated to Host Integration Server or similar programs, less memory is available for file sharing.

Windows Server 2003 or Windows 2000 Server allow you to view or change network throughput options. However, Host Integration Server installation automatically sets the option to maximize throughput for network applications.

Servers used primarily for communications run many important background processes (processes not related to user actions in the current window). These servers generally do not need to run foreground processes at maximum speed. Host Integration Server throughput can be increased by making the operating system more responsive to background processes and less responsive to foreground processes.

A server that is less responsive to foreground processes will run local applications such as word-processing software, spreadsheets, or SNA Manager more slowly. Tasking is most appropriate for servers used primarily to support client systems, and not for servers used locally as desktop computers.

To optimize background processing for a computer running Windows Server 2003 or Windows 2000 Server

1. Click **Start**, point to **Settings**,click **Control Panel**, and then double-click **System**.
2. Select the **Advanced** tab.
3. In the **Performance** box, click **Performance Options**.
4. In the **Application response** box, **select Applications or Background services**.
5. Click **OK**, and then click **OK** again.

See Also
Other Resources
Transaction Programs

APPC Mode

A mode is associated with an LU-LU pair, and determines the session properties for that pair. One of the key mode properties is the Parallel Session Limit. This limit determines whether an LU-LU pair can perform only one interaction at a time or multiple concurrent interactions.

Parallel sessions are used only with independent APPC. If parallel sessions are to be allowed with an LU-LU pair, the local LU must be independent, and the remote LU in the pair must support parallel sessions.

If the LU-LU pair can have multiple parallel sessions, other mode properties, such as Minimum Contention-Winner Limit, determine to what extent each LU can initiate interactions.

The following table lists the modes provided with Host Integration Server.

Mode name	To be used for
#BATCH	Batch-oriented sessions
#BATCHC	Batch-oriented sessions using compression
#BATCHSC	Batch-oriented sessions that employ a minimal level of routing security
BLANK	Sessions using a default mode name, encoded as eight blank EBCDIC spaces in BIND
#INTER	Interactive sessions
#INTERC	Interactive sessions using compression
#INTERSC	Interactive sessions that employ a minimal level of routing security
QPCSUPP	All sessions with an AS/400 computer
QSERVER	ODBC drivers

See Also
Concepts
APPC Mode Definition

Notes Section

This section contains notes about AS/400 computers, local and remote names, time-outs for 802.2 connections, and adapter and adapter addresses.

Note About AS/400 Computers

To connect to an AS/400 computer, supply both local node and remote node parameters for the Network Name and the Control Point Name. These properties are configured in separate dialog boxes:

- **Local node (server) properties.** In the **Server Properties** dialog box of the computer running Host Integration Server, type the **Control Point Name** and the **Network Name**. The server Network Name must correspond to the RMTNETID parameter in the AS/400 APPC controller description. (The default for the AS/400 computer network ID is typically APPN.) If this value is set to ***SAME**, refer to the Local Network ID parameter in the AS/400 Network Attributes screen (using the **dspneta** command). The server Control Point Name must match the RMTCPNAME parameter in the AS/400 APPC controller description. For 802.2 connections, the AS/400 computer can be configured to automatically create the APPC controller description when the computer running Host Integration Server first connects to the AS/400 computer, by setting the AUTOCRTCTL parameter to ***YES** in the AS/400 Token-Ring or Ethernet line description. However, for SDLC connections, the APPC controller description must be manually created.

  If you specify a CPNAME in **Connection Properties**, it overrides the CPNAME in the **Server Properties** dialog box. The CPNAME in **Server Properties** is the default and will be used unless it is changed in **Connections Properties**.

- **Remote node (connection) properties.** In the **Properties** dialog box for an 802.2 or SDLC connection, type the **Network Name** and **Control Point Name (CP name)** of the AS/400 computer into the **Network Name (Remote Node)** and **Control Point Name (Remote Node)** fields. The AS/400 Network Name and Local Control Point Name can be found on the AS/400 Network Attributes screen using the DSPNETA. For more information about these fields, click Help on the **Properties** dialog box for the connection.

  On an AS/400 computer, the Network Name is specified in the Local network ID parameter in the Network Attributes screen and in the RMTNETID parameter on the APPC controller description. For more information, see your APPN documentation.

Using Remote Node Identifiers

You can use several types of parameters to specify the remote node with which a connection should connect:

Remote address (802.2 connections only)

This parameter is required for 802.2 connections. For outgoing calls, it is used to specify the remote host, peer, or downstream system being called. This parameter is the **Remote Network Address**.

Network Name (Remote Node) and Control Point Name (Remote Node)

As a general guideline, use these parameters if the administrator of the remote host, peer, or downstream system uses them.

These two parameters work together; if either parameter is supplied, the other should also be supplied. These parameters are used only for Format 3 exchange identifications (XIDs), one of two types of exchange identification. Format 3 XIDs are generally used for APPC. If a connection does not use Format 3 XIDs, there is no need to specify these parameters.

Remote Node ID

As a general guideline, use this parameter if the administrator of the remote host, peer, or downstream system uses it. This parameter can be used in Format 0 XIDs or Format 3 XIDs, the two types of identification exchange. **Format 0 XID**s are generally used for 3270 communication and for LUA; **Format 3 XID**s are generally used for APPC.

Note About Local and Remote Names

The **Network Name** and **Control Point Name** for the remote node are different from the **Network Name** and **Control Point Name** for the local node. The remote **Network Name** and **Control Point Name** are specified in the **Connection Properties** dialog box. The local **Network Name** and **Control Point Name** are specified in the **Server Properties** dialog box.

Configuring Multidrop Connections

In a multidrop connection, one primary computer communicates simultaneously with multiple secondary computers. Host

Integration Server supports multidrop connections used for downstream systems on leased SDLC lines.

You can configure up to four multidrop connections on each line, with a computer running Host Integration Server as the primary server on each.

Configure each connection to be used in the multidrop configuration according to the instructions that follow.

To configure multidrop connections

1. Configure a link service for an SDLC line with the **Constant RTS** option not selected.
2. Configure each connection to be used in the multidrop configuration as follows:
    - In the **SDLC Properties** dialog box, select the same link service for each connection.
    - In the **SDLC Properties** dialog box, under **Remote End**, select **Downstream**.
    - Make all the parameters identical for each configuration, except for **Poll Address**, and possibly the **Local Node ID** and **Remote Node ID**. For these parameters, obtain values from the downstream system administrators. Make the **Poll Address** unique for each station.
3. For the primary connection in the multidrop configuration, on the SDLC page of the **SDLC Properties** dialog box, select the **Multidrop Primary** box.

For information about using the configuration dialog boxes, click **Help** in the **SDLC Properties** dialog box.

Note About Time-outs for 802.2

The time-outs for 802.2 (**t1**, **t2**, and **ti**) are derived from basic timer values that are set in the Windows registry. Host Integration Server is designed with the assumption that these registry values will not be changed after they have been set by the DLC (802.2) driver. If these registry values are changed, the choices displayed in the **Response Timeout**, **Receive ACK Timeout**, or **Inactivity Timeout** fields may no longer reflect the actual amounts of time that Host Integration Server waits.

Simplified Facility Data Example

The following simplified example shows how identifiers allow each facility request to be distinguished from the ones that follow:

40AAAA20BB

The example contains two requests. The first request begins with a 4, indicating that after the two-character identifier there are four characters (in this case, four occurrences of the A character). The second request follows, beginning with a 2, indicating that after the two-character identifier there are two characters (two occurrences of the B character).

Standard Facility Data Example

The facility data string in this example initiates a connection using 256-byte packets, a sending window of 7, a receive window of 4, and charges reversed. The facility requests that do this are as follows:

- 42 08 08 for Packet Size
- 43 04 07 for Window Size
- 01 01 for reverse charging

When these requests are concatenated, the following facility data string is created:

4208084304070101

Common Identifier Table

The following table shows commonly used identifiers for facility requests.

Facility	Identifier	Parameters	Meaning

Reverse charging	01	00 01	Reverse charging NOT requested Reverse charging requested
Packet Size	42	0r0s	r = Receive Packet Size s = Send Packet Size
			r and s can have the following values:
			6 for 64-byte packets 7 for 128-byte packets 8 for 256-byte packets 9 for 512-byte packets A for 1024-byte packets
Window Size	43	0r0s	r = Receive Window Size s = Send Window Size r and s can be from 1 through 7. They are usually set equal, making the send and receive windows the same size.

In this table, send and receive operate from the view of the initiator of the connection.

Note About Local Adapter Address

When communicating with the administrator of a remote host, peer, or downstream system, you may need to tell that administrator the address of your local adapter.

To determine the local address of an 802.2 adapter, at the command prompt, type:

net config server

In the resulting display, the address of the local adapter appears in the line labeled **Server is Active On**.

Primary Configuration Server

The primary configuration server is the Host Integration Server computer designated to contain the domain-wide configuration file. The configuration file reflects the Host Integration Server resources for the SNA subdomain, including all Host Integration Server computers, link services, LUs, 3270 users, and so on. There can be only one primary server in the subdomain.

Backup Configuration Server

The backup configuration server is a Host Integration Server computer on which the configuration file is replicated by Host Integration Server. There can be more than one backup server in an SNA subdomain. Host Integration Server will load the copy of the configuration file located on a backup Host Integration Server computer if the primary Host Integration Server computer goes down. In this case, servers and connections can be started and stopped, but the configuration cannot be changed or saved.

Identifiers in Incoming XIDs

When deciding which type of remote node identifier to specify for a connection that accepts incoming calls, it may be helpful to understand how Host Integration Server uses exchange identifications (XIDs) from incoming calls.

When Host Integration Server receives an XID from an incoming call, it looks at the XID for some type of identifier of the remote system that made the call. It compares this identifier, in the order shown in the following list, against identifiers stored in the Host Integration Server configuration. If it finds a match, it accepts the call. If it finds that identifiers are left unspecified (in the configuration or the XID), and the connection is an SDLC connection, Host Integration Server accepts the call, pending further exchange of information. In other cases—when every comparison yields a mismatch, or when identifiers are left unspecified but the connection is 802.2—Host Integration Server rejects the incoming call.

Identifiers are compared in the following order:

1. If the incoming XID is Format 3, Host Integration Server examines the XID for a remote node Network Name and Control Point Name. If these parameters are present in both the Incoming XID and in the Host Integration Server configuration, and they match, the call is accepted. If the parameters are present and do not match, the call is rejected.

2. If the parameters were not available for the preceding step, Remote Node IDs are examined next. (Remote Node IDs may be used in either Format 0 or Format 3 XIDs.) If a Remote Node ID is present in both the Incoming XID and in the Host Integration Server configuration, and they match, the call is accepted. If the parameters are present and do not match, the call is rejected.

3. If the parameters were not available for the preceding steps, for 802.2 connections, remote addresses are examined:

   For 802.2 connections, the Remote Network Address in the Host Integration Server configuration is compared to the address from which the XID was received. If the addresses match, the call is accepted; if not, the call is rejected.

4. For 802.2 connections, if no match is found in any of the preceding steps, the incoming call is rejected.

   For SDLC connections, if no match is found in the preceding steps, but identifiers were left unspecified in the configuration or the XID, the call is accepted, pending further exchange of identifiers. However, if identifiers were not left unspecified and no identifiers match, the call is rejected.

Guidelines for Identifiers for 802.2 Connections

The following table lists guidelines for identifiers for 802.2 connections.

Conn. type	**Allowed call directions**	**Guidelines for identifiers for remote node**
802.2	Outgoing	Use Remote Network Address
802.2	Incoming	Match the identifiers used on the remote system: Remote node Network Name and Control Point Name or Remote Node ID or Remote Network Address
802.2	Both	Use a combination of outgoing plus incoming identifiers

Guidelines for Identifiers for SDLC Connections

The following table lists guidelines for identifiers for SDLC connections.

Conn. type	**Allowed call directions**	**Guidelines for identifiers for remote node**
SDLC	Outgoing	If Host Integration Server supplies the phone number to the modem, specify Dial Data (phone number of remote system)
SDLC	Incoming	Match the identifiers used on the remote system: Remote node Network Name and Control Point Name or Remote Node ID
SDLC	Both	Use a combination of outgoing plus incoming identifiers

Making and Testing a Connection

This section provides the procedures used to make successful connections. Several steps are required to connect your host system to Host Integration Server.

One way to remember the four steps required to make a successful connection is with the acronym **LCLUA** that stands for **L**-Link Service, **C**-Connection, **LU**-LUs and **A**-Adding and Assigning Users.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

For information about configuring Host Integration Server for your enterprise environment, see Configuring Your Enterprise.

In This Section

See Also

Other Resources

IP-DLC Link Service

Important Connection Information

This section includes details about the information required to make a successful connection. Review the connection information section to verify your settings.

In This Section

Making a Connection

Items to Consider for a Successful Connection

Verifying Host Connection Information

Verifying Operating System Connection Information

Verifying Host Integration Server Information

See Also

Other Resources

IP-DLC Link Service

Making a Connection

Complete the following steps to give users and groups access to the mainframe environment, install connections, and verify the installation.

(For information on the new IP-DLC Link Service, see IP-DLC Link Service.)

Install Host Integration Server

- Verify Host connection information.
- Verify Microsoft Windows Server 2003 or Windows 2000 Server configuration information.
- Verify Host Integration Server SNA service configuration information
- Install Host Integration Server.
- Install appropriate link services.

Configure Host Integration Server

- Step 1 (L) Creating and Configuring Link Services
- Step 2 (C) Creating and Configuring Connections
- Step 3 (LU) Creating and Configuring 3270 LUs
- Step 4 (A) Adding and Assigning Users

Test Host Integration Server

- Testing Connections with the 3270 Client
- Testing Connections with the 5250 Client

Items to Consider for a Successful Connection

For a connection to be established successfully, a number of software settings and hardware characteristics must work together.

The following table outlines items to consider when configuring a connection with Host Integration Server.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

Element	Considerations
Host configuration settings must match the connectionandserversettings on the Host Integration Server computer.	**Mainframe node ID settings**: For most mainframes, IDBLK and IDNUM in the physical unit (PU) definition must match the two parts of the Remote Node ID on the Host Integration Server connection. **AS/400 name settings**: For the AS/400 computer, local and remote control point names (CP names) and network names must be matched with corresponding Host Integration Server settings **Addresses**: For 802.2, X.25, or channel connections, you must match the host settings with equivalent settings on the Host Integration Server connection. **BTU length**: For the mainframe, the BTU length is set through MAXDATA in the PU definition. For the AS/400 computer, this is set through MAXFRAME. These should equal the Max BTU Length on the Host Integration Server connection. **Other settings**: With SDLC, the NRZ/NRZI settings on the host must match those on the Host Integration Server connection.
For SDLC and X.25 **Communications hardware**: line, modem (if applicable), and adapter characteristics must match thelink serviceandconnectionsettings on the Host Integration Server computer.	**Speed and duplexing**: For SDLC and X.25, note the speed and duplexing capabilities of the line, modem (or DCE), and adapter, to be sure that they will not be exceeded by the settings in the Host Integration Server link service and connection. Settings for fast transmission or for full duplexing cause greater demands on hardware. The hardware element with the smallest capacity limits the capacity of the entire system.

See Also

Other Resources

Important Connection Information

Verifying Host Connection Information

See Also

Other Resources

IP-DLC Link Service

Important Connection Information

Verifying Operating System Connection Information

Collect and review the required operating system information to ensure your operating environment is set up and configured correctly.

See Also

Other Resources

IP-DLC Link Service

Important Connection Information

Verifying Host Integration Server Information

See Also

Other Resources

IP-DLC Link Service

Important Connection Information

Important Configuration Information

The topics in this section detail important configuration information.

In This Section

Using SNA Manager

How to Open a Subdomain

How to Configure Server Properties

How to Configure SNA Service Properties

See Also

Other Resources

IP-DLC Link Service

Using SNA Manager

SNA Manager provides an interface for configuration and operation of Host Integration Server components. SNA Manager allows flexibility in viewing and managing SNA server subdomains, computers, link services, connections, logical units (LUs), sessions, and users. It integrates the administration of TN3720 service, TN5250 service, Host Print service, Shared Folder Gateway service, and Host Security Integration.

You can view all Host Integration Server computers in an SNA server subdomain and manage multiple subdomains at the same time. This allows for central configuration and administration of all servers in an enterprise. You can remotely configure and manage Host Integration Server computers across all popular protocols, including TCP/IP, IPX/SPX, and Microsoft Networking.

SNA Manager can be installed on any computer running Windows Server 2003 or Windows 2000 Server, enabling management of the entire Host Integration Server network from a single computer. In addition, SNA Manager allows more than one administrator to simultaneously view and manage the same SNA server subdomain.

SNA Manager provides a hierarchical, tree-like view of an SNA server subdomain and all its resources.

In the console tree, the subdomain is presented as a hierarchical collection of resources. At the top of the hierarchy is the subdomain itself, which contains Servers, LU Pools, Configured Users, Workstations APPC Modes, CPI-C Symbolic Names, and Host Security Domains. You can view as much of the subdomain detail as needed by opening and closing folders.

When you click an item in the console tree, the details pane shows resources available to the item. Double-clicking an item that does not contain other objects displays the properties of the item.

Shortcut menus (available by right-clicking an object) allow selection of the appropriate command for a particular object. For example, if you select a server and right-click, a shortcut menu allows you to stop the service, control all other services, insert new resources, and view and configure the properties.

SNA Manager lets you administer more than one SNA subdomain by displaying each domain in a separate window with its own hierarchical arrangement of resources.

SNA Manager also simplifies administrative tasks by providing wizards to step you through many of the more complicated tasks, such as configuring AS/400 connectivity, configuring 3270 display LUs, and creating a range of LUs.

See Also

Tasks

How to Open a Subdomain
How to Configure Server Properties
How to Configure SNA Service Properties

Other Resources

IP-DLC Link Service

How to Open a Subdomain

Before you can configure Host Integration Server components, you must open a subdomain.

The **Start Subdomain** dialog box can be used to change SNA subdomains including:

- Local SNA Subdomain
- Remote SNA Subdomain
- Offline Configuration

To open a subdomain

1. Start SNA Manager.
2. In SNA Manager, right-click **SNA Manager** (top level), and select **Open Subdomain**. The **Start SNA Manager for** dialog box appears.
3. Select the appropriate settings for your subdomain, and click **Finish**.

See Also

Tasks

How to Configure Server Properties

How to Configure SNA Service Properties

Concepts

Using SNA Manager

Other Resources

IP-DLC Link Service

How to Configure Server Properties

The **Server Configuration** dialog box can be used to change Host Integration Server properties, including:

- Server Name
- Subdomain Name
- Active Directory OU Name
- Server Role - Primary or Backup
- Restart - MngAgent and SnaBase

To configure server properties

1. Start SNA Manager.
2. In SNA Manager, right-click the server, and select **Properties**.
3. To change the server name, click **Change** near the top of the **Server Configuration** tab. Note that the server must be offline before you can change the name.
4. To set the Group Identity (Subdomain name and Active Directory), Server Role (Primary or Backup), or Network Transports (TCP/IP is default), click **Change** near the bottom of the **Server Configuration** tab.
5. Make the appropriate changes to your configuration, and then click **OK**.
6. You must save your configuration to keep the properties you set. Right-click the server, and click **Save Configuration**.

See Also

Tasks

How to Open a Subdomain

How to Configure SNA Service Properties

Concepts

Using SNA Manager

Other Resources

IP-DLC Link Service

How to Configure SNA Service Properties

The **SNA Service configuration** dialog box can be used to change SNA service properties, including:

- Server Comment
- Network Name
- Control Point Name

To change properties of SNA service

1. Start SNA Manager.
2. In SNA Manager, right-click **SNA Service**, and select **Properties**.
3. Make appropriate changes.
4. Click **OK** when done.
5. You must save your configuration to keep the properties you set. Right-click the server, and click **Save Configuration**.

See Also

Tasks

How to Open a Subdomain

How to Configure Server Properties

Concepts

Using SNA Manager

Other Resources

IP-DLC Link Service

Step 1 (L) Creating and Configuring Link Services

This section details creating and configuring link services. The maximum number of 802.2 link services that one Host Integration Server computer can support is:

- 64 per network adapter (limited by SAP)
- 255 per SNA server (limited by index from 0x01 to 0xFF)

However, SNA Manager allows only 64 link services per Host Integration Server computer, no matter how many adapters are installed on the server. To configure the number higher than this, use the utility Linkcfg.exe.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Creating Link Services

By default, when Host Integration Server is installed, all the software files required by all standard link services are copied to the server, but the link services themselves are not installed or configured.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

The following sections contain procedures detailing how to create link services using SNA Manager.

In This Section

How to Create the DLC 802.2 Link Service

The following procedure details creating a new DLC 802.2 link service.

To create the DLC 802.2. link service

1. Start SNA Manager.
2. In SNA Manager, right-click the server on which to add the link service, point to **New**,and then click **Link Service**.
3. Select the **DLC 802.2 Link Service**, and click **Add**.
4. Configure the following link service information (if required):
   - Title
   - Adapter
   - Local Service Access Point (SAP)
   - Use Fixed SAP option
   - Allow Link Server to be Distributed option
5. Click **OK**.

   The **Insert Link Service** box will continue to be displayed, allowing you to insert additional link services as needed.
6. After you have finished adding the needed link services, click **Finish**.
7. Right-click **Link Services**, and then click **Save configuration**.

Note
When the link services have been inserted, SNA Manager displays them in the **Link Services** folder.

See Also
Tasks
How to Create the 3270 Demonstration
How to Create the 5250 Demonstration
How to Create the LU1 Print Demonstration
How to Create the LU3 Print Demonstration
Other Resources
IP-DLC Link Service
Creating Link Services

How to Create the 3270 Demonstration

The following procedure details creating the Continuous 3270 Demonstration.

To create the 3270 demonstration

1. Start SNA Manager.
2. In SNA Manager, right-click the server on which to add the link service, point to **New**, and then click **Link Service**.
3. Select **DEMO SDLC Link Service**, and click **Add**.
4. Verify the script file is for the **3270 Continuous Demo** and click **OK**.

   The **Insert Link Service** box will continue to be displayed, allowing you to insert additional link services as needed.
5. After you have finished adding the needed link services, click **Finish**.
6. Right-click **Link Services**, and then click **Save configuration**.

Note
When the link services have been inserted, SNA Manager displays them in the **Link Services** folder.

See Also
Tasks
How to Create the DLC 802.2 Link Service
How to Create the 5250 Demonstration
How to Create the LU1 Print Demonstration
How to Create the LU3 Print Demonstration
Other Resources
IP-DLC Link Service
Creating Link Services

How to Create the 5250 Demonstration

The following procedure details creating the 5250 Demonstration.

To create the 5250 demonstration

1. Start SNA Manager.
2. In SNA Manager, right-click the server on which to add the link service, point to **New**,and then click **Link Service**.
3. Select **DEMO SDLC Link Service**,and click **Add**.
4. Verify the script file is for the **AS 400 Demo** and click **OK**.

   The **Insert Link Service** box will continue to be displayed, allowing you to insert additional link services as needed.
5. After you have finished adding the needed link services, click **Finish**.
6. Right-click **Link Services**, and then click **Save configuration**.

Note
When the link services have been inserted, SNA Manager displays them in the **Link Services** folder.

See Also
Tasks
How to Create the DLC 802.2 Link Service
How to Create the 3270 Demonstration
How to Create the LU1 Print Demonstration
How to Create the LU3 Print Demonstration
Other Resources
IP-DLC Link Service
Creating Link Services

How to Create the LU1 Print Demonstration

The following procedure details creating the LU1 Print Demonstration.

To create the LU1 print demonstration

1. In SNA Manager, right-click the server on which to add the link service, point to **New**,and then click **Link Service**.
2. Select **DEMO SDLC Link Service**, and click **Add**.
3. Verify the script file is for the **LU1 Print Demo** and click **OK**.

   The **Insert Link Service** box will continue to be displayed, allowing you to insert additional link services as needed.
4. After you have finished adding the needed link services, click **Finish**.
5. Right-click **Link Services**, and then click **Save configuration**.

Note
When the link services have been inserted, SNA Manager displays them in the **Link Services** folder.

See Also
Tasks
How to Create the DLC 802.2 Link Service
How to Create the 3270 Demonstration
How to Create the 5250 Demonstration
How to Create the LU3 Print Demonstration
Other Resources
IP-DLC Link Service
Creating Link Services

How to Create the LU3 Print Demonstration

The following procedure details creating the LU3 Print Demonstration.

To create the LU3 print demonstration

1. In SNA Manager, right-click the server on which to add the link service, point to **New**,and then click **Link Service**.
2. Select the **DEMO SDLC Link Service**, and click **Add**.
3. Verify the script file is for the **LU3 Print Demo**, and click **OK**.

   The **Insert Link Service** box will continue to be displayed, allowing you to insert additional link services as needed.
4. After you have finished adding the needed link services, click **Finish**.
5. Right-click **Link Services**, and then click **Save configuration**.

Note
When the link services have been inserted, SNA Manager displays them in the **Link Services** folder.

See Also
Tasks
How to Create the DLC 802.2 Link Service
How to Create the 3270 Demonstration
How to Create the 5250 Demonstration
How to Create the LU1 Print Demonstration
Other Resources
IP-DLC Link Service
Creating Link Services

Configuring Link Services

By default, when Host Integration Server is installed, all the software files required by all standard link services are copied to the server. After the link services are installed, they must be configured for your host environment.

The following sections contain procedures detailing how to configure link services.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

In This Section

How to Configure the DLC 802.2 Link Service

Use the following procedure to configure the DLC 802.2 link service.

To configure the DLC 80.2. link service

1. In SNA Manager, expand the server, and then click **Link Service**.
2. Right-click the DLC 802.2 **(SNADLC1)** link service, and click **Properties**.
3. In **Properties**, click **Configure**.
4. Make configuration changes, and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

See Also

Tasks

How to Configure the 3270 Demonstration
How to Configure the 5250 Demonstration
How to Configure the LU1 Print Demonstration
How to Configure the LU3 Print Demonstration

Other Resources

IP-DLC Link Service
Configuring Link Services

How to Configure the 3270 Demonstration

Use the following procedure to configure the 3270 Demonstration link service.

To configure the 3270 demonstration

1. In SNA Manager, expand the serve,r and then click **Link Service**.
2. Right-click the 3270 Demonstration (**SNADEMO1**) link service, and click **Properties**.
3. In **Properties**, click **Configure**.
4. Make configuration changes, and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

See Also

Tasks

How to Configure the DLC 802.2 Link Service
How to Configure the 5250 Demonstration
How to Configure the LU1 Print Demonstration
How to Configure the LU3 Print Demonstration

Other Resources

IP-DLC Link Service
Configuring Link Services

How to Configure the 5250 Demonstration

Use the following procedure to configure the 5250 (AS/400) Demonstration link service.

To configure the 5250 demonstration

1. In SNA Manager, expand the server, and then click **Link Service**.
2. Right-click the 5250 Demonstration (**SNADEMO2**) link service, and click **Properties**.
3. In **Properties**, click **Configure**.
4. Make configuration changes, and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

See Also

Tasks

How to Configure the DLC 802.2 Link Service
How to Configure the 3270 Demonstration
How to Configure the LU1 Print Demonstration
How to Configure the LU3 Print Demonstration

Other Resources

IP-DLC Link Service
Configuring Link Services

How to Configure the LU1 Print Demonstration

Use the following procedure to configure the LU1 Print Demonstration link service.

To configure the LU1 print demonstration

1. In SNA Manager, expand the server, and then click **Link Service**.
2. Right-click the LU1 Print Demonstration (**SNADEMO3**) link service, and click **Properties**.
3. In **Properties**, click **Configure**.
4. Make configuration changes, and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

See Also

Tasks

How to Configure the DLC 802.2 Link Service
How to Configure the 3270 Demonstration
How to Configure the 5250 Demonstration
How to Configure the LU3 Print Demonstration

Other Resources

IP-DLC Link Service
Configuring Link Services

How to Configure the LU3 Print Demonstration

Use the following procedure to configure the LU3 Print Demonstration link service.

To configure the LU3 print demonstration

1. In SNA Manager, expand the server, and then click **Link Service**.
2. Right-click the LU3 Print Demonstration (**SNADEMO4**) link service, and click **Properties**.
3. In **Properties**, click **Configure**.
4. Make configuration changes, and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

See Also

Tasks

How to Configure the DLC 802.2 Link Service
How to Configure the 3270 Demonstration
How to Configure the 5250 Demonstration
How to Configure the LU1 Print Demonstration

Other Resources

IP-DLC Link Service
Configuring Link Services

Step 2 (C) Creating and Configuring Connections

This section details creating and configuring connections.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Creating Connections

Configuring Connections

Creating Connections

With Host Integration Server, you can create connections with wizards or manually. Host Integration Server provides you with three wizards to aid you in installing and configuring connections to your host environment (HE).

These tools step you through configuring connection properties, creating 3270 display LUs and an LU pool, and assigning the LUs to the LU pool.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Mainframe (3270) environments:

How to Create a 3270 Connection Using a Wizard

How to Create a 3270 Connection Manually

AS/400 (5250) environments:

How to Create a 5250 Connection Using a Wizard

How to Create a 5250 Connection Manually

How to Create a 5250 Local APPC LU

How to Create a 5250 Remote APPC LU

How to Create a 3270 Connection Using a Wizard

The following procedure details using a wizard to create a 3270 connection. The 3270 Continuous link service will be used as an example.

To create a 3270 connection using a wizard

1. In SNA Manager, right-click the server on which to add a connection.
2. Point to **All Tasks**, and then select the **3270 Wizard** option.
3. Click **Next** on the **Welcome** screen.
4. Select the computer where SNA service resides, and click **Next**.
5. Enter a **Name** for the connection, **DM3270**, and click **Next**.
6. Select the appropriate link service, **SNADEMO1**, and click **Next**.
7. Accept the default Local Node ID information, and click **Next**.
8. Accept the default PU address information, and click **Next**.
9. Accept the default LU information, and click **Next**.
10. Click **Add** to add a member (user).
11. Select a member, click **Add**, and then click **OK**.
12. Click **Next**.
13. Click **Finish** to complete the wizard.
14. Click **OK** on the final wizard information page.
15. When the wizard completes, right-click **SNA Service**, and click **Save configuration**.
16. Stop and then start SNA service.

See Also

Tasks

How to Create a 3270 Connection Manually
How to Create a 5250 Connection Using a Wizard
How to Create a 5250 Connection Manually
How to Create a 5250 Local APPC LU
How to Create a 5250 Remote APPC LU

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

How to Create a 3270 Connection Manually

The following procedure details creating a 3270 connection manually.

To create a 3270 connection manually

1. In SNA Manager, expand the server on which to create the connection, and then expand **SNA Service**.
2. Right-click **Connections**, point to **New**, and then click the type of connection (802.2) to be created.
3. Configure the **Connection Properties**. You are required to specify which link service to use for the connection, as well as the name of the connection and additional information. The choices you make depend on the purpose of the connection.

   The following information must be configured correctly to make a connection:

   - General Tab
   - Address Tab
   - System Identification Tab
   - DLC 802.2 Tab

4. Click **OK**.
5. Right-click **SNA Service**, and then click **Save configuration**.
6. Stop and then start SNA service.

To display this dialog box after the connection has been created, double-click the connection in the tree view, or select the connection, and click **Properties** in the **View** menu.

See Also

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

How to Create a 5250 Connection Using a Wizard

The following procedure details using a wizard to create a 5250 connection.

To create a 5250 connection using a wizard

1. In SNA Manager, right-click the server on which to add a connection.
2. Point to **All Tasks**, and then select the **AS/400 Wizard** option.
3. Click **Next** on the **Welcome** screen.
4. Select the computer where SNA service resides, and click **Next**.
5. Enter a name for the connection, **DM5250**, and click **Next**.
6. Select the appropriate link service, **SNADEMO2**, and click **Next**.
7. Accept the default network name and control point name, and click **Next**.
8. Accept the default PU address information, and click **Next**.
9. Click **Finish** to complete the wizard.
10. Click **OK** on the final wizard information dialog box.
11. When the wizard completes, right-click **SNA Service**, and click **Save configuration**.
12. Stop and then start SNA service.

See Also

Tasks

How to Create a 3270 Connection Using a Wizard
How to Create a 3270 Connection Manually
How to Create a 5250 Connection Manually
How to Create a 5250 Local APPC LU
How to Create a 5250 Remote APPC LU

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

How to Create a 5250 Connection Manually

The following procedure details creating a 5250 connection manually.

To create a 5250 connection manually

1. In SNA Manager, expand the server on which to create the connection, and then expand **SNA Service**.
2. Right-click **Connections**, point to **New**, and then click the type of connection (such as SDLC or 802.2) to be created.

   Connection Properties lets you specify which link service to use for the connection, as well as the name of the connection and additional information. The choices you make depend on the purpose of the connection.
3. Click **OK**.
4. Right-click **SNA Service**, and then click **Save configuration**.
5. Stop and then start SNA service.

   To display this dialog box after the connection has been created, double-click the connection in the tree view. Or select the connection and then, on the **View** menu, click **Properties**.

See Also

Tasks

How to Create a 3270 Connection Using a Wizard
How to Create a 3270 Connection Manually
How to Create a 5250 Connection Using a Wizard
How to Create a 5250 Local APPC LU
How to Create a 5250 Remote APPC LU

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

How to Create a 5250 Local APPC LU

The following procedure details creating a 5250 local APPC LU manually.

To create a 5250 local APPC LU

1. In SNA Manager, expand the server on which to add the **Local APPC LU**.
2. Expand **SNA Service**, and right-click **Local APPC LU**.
3. Point to **New**, and then select **Local LU**.
4. On the **General** tab:
    - Enter LU Alias.
    - Enter Network Name.
    - Enter LU Name.
    - Enter Comment (optional).
5. On the **Advanced** tab:
    - Select if this is a member of default local APPC LU pool.
    - Enter time-out in seconds for starting invokable TPs.
    - Select implicit incoming remote LU.
    - Select LU 6.2 type.
    - Select SyncPoint support.
6. Click **OK**.
7. Right-click **SNA Service**, and click **Save configuration**.
8. Stop and then start SNA service.

See Also

Tasks

How to Create a 3270 Connection Using a Wizard
How to Create a 3270 Connection Manually
How to Create a 5250 Connection Using a Wizard
How to Create a 5250 Connection Manually
How to Create a 5250 Remote APPC LU

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

How to Create a 5250 Remote APPC LU

The following procedure details creating a 5250 remote APPC LU manually.

To create a 5250 remote APPC LU

1. In SNA Manager, expand the server on which to add the **Remote APPC LU**.
2. Expand **SNA Service**, and right-click **Remote APPC LU**.
3. Point to **New**, and then select **Remote LU**.
4. On the **General** tab:
    - Enter Connection.
    - Enter LU Alias.
    - Enter Network Name.
    - Enter LU Name.
    - Enter Uninterpreted Name.
    - Enter Comment (optional).
5. On the **Options** tab:
    - Select if this remote LU supports parallel sessions.
    - Select implicit incoming remote LU.
    - Select Session-Level security.
    - Select SyncPoint support.
6. Click **OK**.
7. Right-click **SNA Service**, and click **Save configuration**.
8. Stop and then start SNA service.

See Also

Tasks

How to Create a 3270 Connection Using a Wizard
How to Create a 3270 Connection Manually
How to Create a 5250 Connection Using a Wizard
How to Create a 5250 Connection Manually
How to Create a 5250 Local APPC LU

Concepts

Creating Connections

Other Resources

IP-DLC Link Service

Configuring Connections

If you run a wizard to create your connections, most of the configuration information will be added with the criteria you entered when the wizard runs. The AS/400 Wizard, the Mainframe APPC/LU6.2 Wizard, and the 3270 Wizard step you through configuring connection properties, creating 3270 display LUs and an LU pool, and assigning the LUs to the LU pool.

If you need to configure a connection manually, you may need to verify the host connection information before you can proceed. For more information, see Verifying Host Connection Information.

For information on the new IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Mainframe (3270) environments

How to Configure a 3270 Connection

AS/400 (5250) environments

How to Configure a 5250 Connection

How to Configure a 5250 Local APPC LU

How to Configure a 5250 Remote APPC LU

How to Configure a 3270 Connection

The following procedure details configuring a 3270 connection.

To configure a 3270 connection

1. In SNA Manager, expand the server on which to configure the connection, and then expand **SNA Service**.
2. Expand **Connections**, right-click the 3270 (**DM3270**) connection, and then click **Properties**.

   Connection Properties lets you specify which link service to use for the connection, as well as the name of the connection and additional information. The choices you make depend on the purpose of the connection.
3. After you have made your configuration changes, click **OK**.
4. Right-click **SNA Service**, and then click **Save configuration**.
5. Stop and then start SNA service.

See Also
Tasks
How to Configure a 5250 Connection
How to Configure a 5250 Local APPC LU
How to Configure a 5250 Remote APPC LU
Concepts
Configuring Connections
Other Resources
IP-DLC Link Service

How to Configure a 5250 Connection

The following procedure details configuring a 5250 connection.

To configure a 5250 connection

1. In SNA Manager, expand the server on which to configure the connection, and then expand **SNA Service**.
2. Expand **Connections**, right-click the 5250 (**DM5250**) connection, and then click **Properties**.

   Connection Properties lets you specify which link service to use for the connection, as well as the name of the connection and additional information. The choices you make depend on the purpose of the connection.
3. After you have made your configuration changes, click **OK**.
4. Right-click **SNA Service**, and then click **Save configuration**.
5. Stop and then start SNA service.

See Also
Tasks
How to Configure a 3270 Connection
How to Configure a 5250 Local APPC LU
How to Configure a 5250 Remote APPC LU
Concepts
Configuring Connections
Other Resources
IP-DLC Link Service

How to Configure a 5250 Local APPC LU

The following procedure details configuring a 5250 local APPC LU.

To configure a 5250 local APPC LU

1. In SNA Manager, expand the server on which to configure the connection, and then expand **SNA Service**.
2. Expand **Local APPC LUs**, right-click the appropriate 5250 local APPC LU, and then click **Properties**.

   Connection Properties lets you specify which LU alias to use, as well as the name of the network and LU name information. The choices you make depend on the purpose of the connection.
3. After you have made your configuration changes, click **OK**.
4. Right-click **SNA Service**, and then click **Save configuration**.
5. Stop and then start SNA service.

See Also
Tasks
How to Configure a 3270 Connection
How to Configure a 5250 Connection
How to Configure a 5250 Remote APPC LU
Concepts
Configuring Connections
Other Resources
IP-DLC Link Service

How to Configure a 5250 Remote APPC LU

The following procedure details configuring a 5250 remote APPC LU.

To configure a 5250 remote APPC LU

1. In SNA Manager, expand the server on which to configure the connection, and then expand **SNA Service**.
2. Expand **Remote APPC LUs**, right-click the appropriate 5250 remote APPC LU, and then click **Properties**.

   Connection Properties lets you specify which LU alias to use, as well as the name of the network and LU name information. The choices you make depend on the purpose of the connection.
3. After you have made your configuration changes, click **OK**.
4. Right-click **SNA Service**, and then click **Save configuration**.
5. Stop and then start SNA service.

See Also
Tasks
How to Configure a 3270 Connection
How to Configure a 5250 Connection
How to Configure a 5250 Local APPC LU
Concepts
Configuring Connections
Other Resources
IP-DLC Link Service

Step 3 (LU) Creating and Configuring 3270 LUs

This section details information regarding creating and configuring LUs.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Creating LUs

After configuring the connection, you can create the 3270 LUs. The LUs can be display (terminal emulation) LUs, printer LUs, application LUs (LUAs), or downstream LUs.

Each 3270 LU is configured based on the type of connection used. If the connection is channel, 802.2, X.25, or SDLC, the LUs need an LU number, which identifies the LU on its connection. This number should match the LOCADDR= parameter of the LU definition in Virtual Telecommunications Access Method (VTAM) or in the Network Control Program (NCP) GEN. Up to 254 LUs can be configured for each connection, and they can be consecutively configured as a range of LUs.

You can create LUs one at a time or in a consecutively numbered range. When creating a range of LUs, all the LUs are given the same properties. You can modify individual LUs after creating them.

In This Section

How to Create a 3270 Display LU

The following procedure details how to create a display LU.

To create a 3270 display LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**, point to **New**, and then click **Display LU**.

   You can define LUs in this **Properties** dialog box. Keep the default for the **LU Number** box as 2. Two is the first available number, because one is reserved for the host.

   The LU number is meaningful to the connection. You can add a user-friendly name in the **LU Name** box.
3. Type the LU Name, **LU 2**. Leave the other boxes as they are.
4. Click the **Display Model** tab and choose **2 (24 X 80)** as the display station model. Because models can change without notification, click **Model can be overridden**.
5. Click the **Associated Printer** tab. When configuring an LU, these boxes are generally left blank.
6. Click **OK**.

See Also

Tasks

How to Create a 3270 Printer LU
How to Create a 3270 Application LU (LUA)
How to Create a 3270 Downstream LU
How to Create a Local APPC LU
How to Create a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Create a 3270 Printer LU

The following procedure details how to create a printer LU.

To create a 3270 printer LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**, point to **New**, and then click **Printer LU**.

   You can define LUs in this **Properties** dialog box. Keep the default for the **LU Number** box as 3. Three is the first available number, because two was used in the previous example and one is reserved for the host.

   The LU number is meaningful to the connection. You can add a user-friendly name in the **LU Name** box.
3. Type the LU Name, **LU 4**. (You might have expected LU 3, but LU 3 is reserved for the LU3 demo.) Leave the other boxes as they are.
4. Type a comment (optional).
5. If you are using compression, select the **Use Compression** box.
6. If the workstation is secured, select the **User Workstation Secured** box.
7. Click **OK**.

See Also

Tasks

How to Create a 3270 Display LU
How to Create a 3270 Application LU (LUA)
How to Create a 3270 Downstream LU
How to Create a Local APPC LU
How to Create a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Create a 3270 Application LU (LUA)

The following procedure details how to create an application LU (LUA).

To create a 3270 application LU (LUA)

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**, point to **New**, and then click **Application LU (LUA)**.

   You can define LUs in this **Properties** dialog box. Keep the default for the **LU Number** box as 4. Four is the next available number, because two and three were used in the previous examples, and one is reserved for the host.

   The LU number is meaningful to the connection. You can add a user-friendly name in the **LU Name** box.
3. Type the LU Name, **LU 5**. Leave the other boxes as they are.
4. Type a comment (optional).
5. If you are using compression, select the **Use Compression** box.
6. If the workstation is secured, select the **User Workstation Secured** box.
7. If you require high priority mode, select the **High Priority Mode** box.
8. Click **OK**.

See Also

Tasks

How to Create a 3270 Display LU
How to Create a 3270 Printer LU
How to Create a 3270 Downstream LU
How to Create a Local APPC LU
How to Create a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Create a 3270 Downstream LU

The following procedure details how to create a downstream LU.

To create a 3270 downstream LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**, point to **New**, and then click **Downstream LU**.

   You can define LUs in this **Properties** dialog box. Keep the default for the **LU Number** box as 5. Five is the next available number, because two, three, and four were used in the previous examples and one is reserved for the host.

   The LU number is meaningful to the connection. You can add a user-friendly name in the **LU Name** box.
3. Type the LU Name, **LU 6**. Leave the other boxes as they are.
4. Type a comment (optional).
5. Click **OK**.

See Also

Tasks

How to Create a 3270 Display LU
How to Create a 3270 Printer LU
How to Create a 3270 Application LU (LUA)
How to Create a Local APPC LU
How to Create a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Create a Local APPC LU

The following procedure details how to create a local APPC LU.

To create a local APPC LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Local APPC LUs**.
2. Right-click **Local APPC LUs**, point to **New**, and then click **Local LU**.
3. On the **General** tab:
    - Type the LU Alias.
    - Type the Network Name.
    - Type the LU Name.
    - Type a comment (optional).
4. On the **Advanced** tab:
    - If you are a member of the local APPC LU pool, select the **Member of Default Outgoing Local APPC LU Pool** box.
    - Type the amount of time in seconds for **Timeout for Staring Invokable TPs**.
    - Select the **Implicit Incoming Remote LU**.
    - Select the **LU 6.2 Type (Independent or Dependent)**.
    - Select **SyncPoint Support** if required.
5. Click **OK**.

See Also

Tasks

How to Create a 3270 Display LU
How to Create a 3270 Printer LU
How to Create a 3270 Application LU (LUA)
How to Create a 3270 Downstream LU
How to Create a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Create a Remote APPC LU

The following procedure details how to create a remote LU.

To create a remote APPC LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Remote APPC LUs**.
2. Right-click **Remote APPC LUs**, point to **New**, and then click **Remote LU**.
3. On the **General** tab:
    - Type the LU Alias.
    - Type the Network Name.
    - Type the LU Name.
    - Type the Uninterpreted Name.
    - Type a comment (optional).
4. On the **Options** tab:
    - Select the **Support Parallel Sessions** box.
    - Select the **Implicit Incoming Mode**.
    - Select the **Session-Level Security** mode.
    - Select **SyncPoint Support** if required.
5. Click **OK**.

See Also

Tasks

How to Create a 3270 Display LU
How to Create a 3270 Printer LU
How to Create a 3270 Application LU (LUA)
How to Create a 3270 Downstream LU
How to Create a Local APPC LU

Other Resources

IP-DLC Link Service

Configuring LUs

This section provides information detailing configuring logical units (LU).

In This Section

How to Configure a 3270 Display LU

The following procedure details how to configure a display LU.

To configure a 3270 display LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**. You should see the available LUs in the details pane.
3. Click the **Display LU**, **LU 2**, and then click **Properties**.
4. Make appropriate configuration changes, and then click **OK**.

See Also

Tasks

How to Configure a 3270 Printer LU
How to Configure a 3270 Application LU (LUA)
How to Configure a 3270 Downstream LU
How to Configure a Local APPC LU
How to Configure a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Configure a 3270 Printer LU

The following procedure details how to configure a printer LU.

To configure an 3270 printer LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**. You should see the available LUs in the details pane.
3. Click the **Printer LU**, **LU 3**, and then click **Properties**.
4. Make appropriate configuration changes, and then click **OK**.

See Also

Tasks

How to Configure a 3270 Display LU
How to Configure a 3270 Application LU (LUA)
How to Configure a 3270 Downstream LU
How to Configure a Local APPC LU
How to Configure a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Configure a 3270 Application LU (LUA)

The following procedure details how to configure an application LU (LUA).

To configure a 3270 application LU (LUA)

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**. You should see the available LUs in the details pane.
3. Click the **Application LU**, **LU 4**, and then click **Properties**.
4. Make appropriate configuration changes, and then click **OK**.

See Also

Tasks

How to Configure a 3270 Display LU
How to Configure a 3270 Printer LU
How to Configure a 3270 Downstream LU
How to Configure a Local APPC LU
How to Configure a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Configure a 3270 Downstream LU

The following procedure details how to configure a downstream LU.

To configure a 3270 downstream LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Connections**.
2. Right-click the appropriate connection, **DM3270**. You should see the available LUs in the details pane.
3. Click the **Downstream LU**, **LU 5**, and then click **Properties**.
4. Make appropriate configuration changes, and then click **OK**.

See Also

Tasks

How to Configure a 3270 Display LU
How to Configure a 3270 Printer LU
How to Configure a 3270 Application LU (LUA)
How to Configure a Local APPC LU
How to Configure a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Configure a Local APPC LU

The following procedure details how to configure a local APPC LU.

To configure a local APPC LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Local APPC LUs**.
2. Right-click the appropriate **Local APPC LU**, and then click **Properties**.
3. On the **General** tab:
   - Verify LU Alias.
   - Verify Network Name.
   - Verify LU Name.
4. On the **Advanced** tab:
   - Verify you are a member of the local APPC LU pool, and then select the **Member of Default Outgoing Local APPC LU Pool** box.
   - Verify the amount of time in seconds for **Timeout for Staring Invokable TPs**.
   - Verify the **Implicit Incoming Remote LU**.
   - Verify the **LU 6.2 Type**.
   - Verify **SyncPoint Support**.
5. Click **OK**.

See Also

Tasks

How to Configure a 3270 Display LU
How to Configure a 3270 Printer LU
How to Configure a 3270 Application LU (LUA)
How to Configure a 3270 Downstream LU
How to Configure a Remote APPC LU

Other Resources

IP-DLC Link Service

How to Configure a Remote APPC LU

The following procedure details how to configure a remote LU.

To configure a remote APPC LU

1. In SNA Manager, expand **Servers**, expand **SNA Service**, and then expand **Local APPC LUs**.
2. Right-click the appropriate remote LU, and then click **Properties**.
3. On the **General** tab:
    - Verify Connection.
    - Verify Network Name.
    - Verify LU Name.
    - Verify Uninterpreted Name.
4. On the **Options** tab:
    - Verify Supports Parallel Sessions box.
    - Verify Implicit Incoming Mode.
    - Verify Session-Level Security.
    - Verify SyncPoint Support.
5. Click **OK**.

See Also

Tasks

How to Configure a 3270 Display LU
How to Configure a 3270 Printer LU
How to Configure a 3270 Application LU (LUA)
How to Configure a 3270 Downstream LU
How to Configure a Local APPC LU

Other Resources

IP-DLC Link Service

Step 4 (A) Adding and Assigning Users

This section describes adding users and assigning LUs to configured users.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

In This Section

How to Add New Users

Although Host Integration Server uses the User accounts of the Windows Server 2003 or Windows 2000 Active Directory directory service, users must be configured to use Host Integration Server resources such as LUs.

To add new users

1. In SNA Manager, expand the SNA subdomain where the users reside.
2. Right-click **Configured Users**, point to **New**, and then click **User**.
3. Select a user (or group) from the list, and click **OK**.

See Also

Tasks

How to Assign LUs to Configured Users

How to Assign Remote APPC LUs to Configured Users

Other Resources

IP-DLC Link Service

How to Assign LUs to Configured Users

Although Host Integration Server uses the User accounts of the Windows Server 2003 or Windows 2000 Active Directory directory service, users must be configured to use Host Integration Server resources such as LUs.

To assign LUs to configured users

1. In SNA Manager, expand the SNA subdomain where the users reside.
2. Expand **Configured Users**. Select a user, and then right-click **Assign LUs**.
3. Select an LU, and then click **OK**.

The list of LUs that a particular user can access depends on how the user connects to Host Integration Server. If Active Directory is used, a user can access any LUs assigned directly to the user account and can also access any LUs assigned to any of the groups the user is a member of. If not using Active Directory, a user can only access LUs assigned directly to the user, or if no LUs are assigned to the user account, then the user can access LUs assigned to exactly one group. The groups are checked in the following order: global groups, local groups, well known groups. There is no ordering within a group.

See Also

Tasks

How to Add New Users

How to Assign Remote APPC LUs to Configured Users

Other Resources

IP-DLC Link Service

How to Assign Remote APPC LUs to Configured Users

Although Host Integration Server uses the User accounts of the Windows Server 2003 or Windows 2000 Active Directory directory service, users must be configured to use Host Integration Server resources such as LUs.

To assign remote APPC LUs to a configured user

1. In SNA Manager, expand the SNA subdomain where the users reside.
2. Select **Servers**, expand **Servers**, select a server, expand it, and then expand **Remote APPC LUs**.
3. Select an LU, right-click it, and assign it to a user.
4. Select the user, and click **OK**.

See Also

Tasks

How to Add New Users

How to Assign LUs to Configured Users

Other Resources

IP-DLC Link Service

Testing Connections

The topics in this section detail testing, including running and configuring, the 3270 and 5250 Client applications.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Testing Connections with the 3270 Client

Testing Connections with the 5250 Client

Testing Connections with the 3270 Client

The 3270 Client can be used to access your mainframe (3270) environment.

In This Section

How to Run the 3270 Client

The following procedure details running the 3270 Client application.

To run the 3270 client

1. Start **SNA Manager**.
2. On the **Tools** menu, click **3270 Client**. This launches the Host Integration Server 3270 Client application.
3. Next, configure the 3270 Client application. For details, see How to Configure the 3270 Client.

How to Configure the 3270 Client

The following procedure details configuring the 3270 Client application.

The 3270 continuous demo will be used for this procedure.

To configure the 3270 client

1. Start SNA Manager.
2. On the **Tools** menu, click **3270 Client**.
3. On the **Session** menu, click **Session Configuration**.
4. Select the correct LU or pool name, **DM3270**, and then click **OK**.
5. Next, run the 3270 demonstration. For details, see Running the 3270 Client Demonstration.

How to Run the 3270 Client Demonstration

The following procedure details running the 3270 continuous demonstration.

To run the 3270 client demonstration

1. Start SNA Manager.
2. On the **Tools** menu, click **3270 Client**.
3. On the **Session** menu, click **Connect**, which connects you to the 3270 demonstration script.
4. Return to SNA Manager. The status of the DM3270 connection should show **Active**.

See Also

Other Resources

IP-DLC Link Service

Testing Connections with the 3270 Client

Testing Connections with the 5250 Client

The 5250 Client can be used to access your AS/400 environment. The following topics detail running and configuring the 5250 Client application.

In This Section

How to Run the 5250 Client

The following procedure details running the 3270 Client application.

To run the 5250 client

1. Start SNA Manager.
2. Right-click **SNA Service**, and click **Start**.
3. On the **Tools** menu, click **5250 Client**. This launches the Host Integration Server 3270 Client application.
4. Next, configure the 5250 Client application. For details, see Configuring the 5250 Client.

How to Configure the 5250 Client

The following procedure details configuring the 5250 Client application.

The 5250 (AS/400) demo will be used for this procedure.

To configure the 5250 client

1. Start SNA Manager.
2. On the **Tools** menu, click **5250 Client**.
3. On the **Session** menu, click **Session Configuration**.
4. Select the correct LU or pool name, **DM5250**, and then click **OK**.
5. Next, run the 5250 demonstration. For details, see Running the 5250 Client Demonstration.

How to Run the 5250 Client Demonstration

The following procedure details running the 5250 demonstration.

To run the 5250 client demonstration

1. Start SNA Manager.
2. On the **Tools** menu, click **5250 Client**.
3. On the **Session** menu, click **Connect**, which connects you to the 5250 (AS/400) demonstration script.
4. Return to SNA Manager. The status of the DM5250 connection should show **Active**.

See Also

Other Resources

Testing Connections with the 5250 Client

Configuring Your Enterprise

Making and Testing a Connection provides the basic steps to configure a link service, make a connection, create logical units (LU), and log on to a host computer; this section follows a similar format and provides the concepts and procedures for configuring Host Integration Server in your enterprise.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

In This Section

Creating and Configuring Link Services

By default, when Host Integration Server is installed, all the software files required by all standard link services are copied to the server, but the link services themselves are not installed or configured.

For information about the IP-DLC Link Service, see IP-DLC Link Service.

The following procedures detail how to create link services using SNA Manager:

- How to Create the DLC 802.2 Link Service
- How to Create the 3270 Demonstration
- How to Create the 5250 Demonstration
- How to Create the LU1 Print Demonstration
- How to Create the LU3 Print Demonstration

The following procedures detail how to configure link services using SNA Manager:

- How to Configure the DLC 802.2 Link Service
- How to Configure the 3270 Demonstration
- How to Configure the 5250 Demonstration
- How to Configure the LU1 Print Demonstration
- How to Configure the LU3 Print Demonstration

Creating and Configuring Connections

You can create connections between Host Integration Server and your host computer with wizards or manually. Host Integration Server provides three wizards to help you install and configure connections to your host environment (HE).

The AS/400 Wizard, the Mainframe APPC/LU 6.2 Wizard, and the 3270 Wizard step you through configuring connection properties, creating 3270 display LUs and an LU pool, and assigning the LUs to the LU pool.

For information about the new IP-DLC Link Service, see IP-DLC Link Service.

The following procedures detail how to create connections:

Mainframe (3270) environments:

- Creating a 3270 Connection Using a Wizard
- Creating a 3270 Connection Manually

AS/400 (5250) environments:

- Creating a 5250 Connection Using a Wizard
- Creating a 5250 Connection Manually
- Creating a 5250 Local APPC LU
- Creating a 5250 Remote APPC LU

If you use a wizard to create your connections, most of the configuration information is added along with the criteria you entered when the wizard runs.

If you need to configure a connection manually, you may need to verify some of your connection information. For more information, see Important Connection Information.

Service Connection Point

During installation, a Service Connection Point in Active Directory is created by Host Integration Server 2009 that identifies this instance of the product. The Active Directory Schema defines a **serviceConnectionPoint** (SCP) object class to make it easy for a service to publish service-specific data in the directory. Users can search the Global Catalog for all instances of the product's Service Connection Point. When the Host Integration Server 2009 is uninstalled, the Service Connection Point is removed from Active Directory.

Service Connection Point properties

Property	Value\Description
serviceClassName	HISServer (matching the SPN used for Kerberos)
serviceBindingInformation	Name of the HIS Subdomain (blank if SNA Gateway is not installed)
serviceDNSNameType	'A' (meaning host type)
serviceDNSName	NETBIOS Host name
Keywords	List of words to identify this server.

Keyword list

Keyword	Description
"Host Integration Server 2009"	Product Name
"Microsoft"	Company Name
"8.0"	Product Version
{3CA45AFD-4759-4768-9BA2-FA516043DA34}	SNA Gateway
{39707A81-0933-453f-8D90-6A0CFA851D94}	Data Providers
{7609DE49-9AAC-41f0-A606-9BA2D69012A0}	Transaction Integrator
{80B96EA7-CD92-4067-BC4A-AC59F1B87602}	MSMQ-MQSeries Bridge
{52163C3B-D6D5-4c71-A768-DA581AA5D632}	Session Integrator

Creating and Configuring LUs

After you configure the connection, you can create the 3270 LUs. The LUs can be display LUs (terminal emulation), printer LUs, application LUs (LUAs), or downstream LUs.

Each 3270 LU is configured based on the type of connection used. If the connection is channel, 802.2, X.25, or Synchronous Data Link Control (SDLC), the LUs need an LU number, which identifies the LU on its connection. This number should match the LOCADDR= parameter of the LU definition in Virtual Telecommunications Access Method (VTAM) or in the Network Control Program (NET) Gen. Up to 254 LUs can be configured for each connection, and they can be consecutively configured as a range of LUs.

You can create LUs one at a time or in a consecutively numbered range. When you create a range of LUs, all the LUs are given the same properties. You can modify individual LUs after creating them.

For information about the new IP-DLC Link Service, see IP-DLC Link Service.

For procedures about creating LUs, see the following:

- Creating a 3270 Display LU
- Creating a 3270 Printer LU
- Creating a 3270 Application LU (LUA)
- Creating a 3270 Downstream LU
- Creating a Local APPC LU
- Creating a Remote APPC LU

For procedures about configuring LUs, see the following:

- Configuring a Range of 3270 LUs
- Assigning LUs to Workstations
- Associating 3270 Printer LUs with 3270 Display LUs
- Configuring Downstream Connections
- Configuring APPC LUs for TPs or 5250 Emulation
- Precedence of Accounts in Determining Default LUs

To modify a 3270 LU

1. In SNA Manager, ensure that the appropriate service is inactive by right-clicking **SNA service**, and then click **Stop**.
2. Double-click **Connections**, and then double-click the connection that contains the LU you want to view.
3. Right-click the LU whose properties you want to view, and then click **Properties**. Click the appropriate tab to view or modify the LU properties.
4. Click **Save Configuration** on the **Action** menu if you made changes, and then restart the affected service.

Note
Restarting the server disconnects all users. Try to schedule configuration changes when they least affect users.

How to Configure a Range of 3270 LUs

You can create a consecutively numbered range of logical units (LU). To do so, you must identify the base LU name limited to eight characters on which the numbered range of names will be built.

After identifying the base LU name, you can create the LUs by identifying the number of LUs in the range and the number the range will start with. Host Integration Server will create LUs using the sequential numbering scheme.

To configure a range of 3270 LUs

1. In SNA Manager, ensure that the appropriate service is inactive by right-clicking **SNA service**, and then click **Stop**.
2. Right-click the connection to which you want to assign the LUs. Point to **All Tasks**, and then click the **Range of LUs**.
3. Fill in the LU Creation Wizard, and click **Finish** when done.
4. On the **Action** menu, click **Save Configuration**.
5. On the **Action** menu, click **Refresh**.

Note
When you create a range of LUs, all the LUs are given the same properties. You can modify individual LUs after creating them.

See Also

Tasks

Creating and Configuring LUs

How to Create LU Pools

Although you can create individual LUs and assign them to users and groups, using LU pools to manage and deploy a large number of LUs is a more efficient method of administering these resources.

LU pools are groupings of 3270, LUA, or downstream LUs that allow you to maximize access to LUs. A user, LUA application, or downstream system using the pool can get LU access as long as one of the pooled LUs is available. If some of the pooled LUs are not functioning, the user, LUA application, or downstream system will not be affected, as long as another LU in the pool is available.

LU pools allow groups of intermittent users to use resources efficiently. For example, ten 3270 users who only need intermittent access may find that a pool of five 3270 LUs is adequate for their needs.

If a user needs access to multiple LU sessions, you can assign one LU or LU pool for each session that the user needs. A pool can be assigned multiple times to the same user.

An LU can be removed from an LU pool by simply deleting it from the pool.

The following procedures will help you manage your LU pools:

- To create LU pools
- To assign a 3270 LU to a connection
- To assign LUs and LU pools to users and groups
- To view LUs or pools assigned to users
- To add 3270 LUs to a pool
- To add a workstation and assign LUs
- To remove an LU from a pool

To create LU pools

1. In SNA Manager, right-click the **Pools** folder, point to **New**, and then click **3270 Display LU Pool**.
2. In **Pool Properties**, enter a name for your pool, up to eight characters in length. A comment is optional.
3. Leave the check box cleared, and click **OK**.
4. The new pool appears in the **Pools** folder.

To assign a 3270 LU to a connection

1. In SNA Manager, expand SNA service for the server you are working with, and then expand **Connections**.
2. Right-click the connection where you want to add the LU, point to **New**, and click the type of 3270 LU you want to assign.
3. In the **3270 LU Properties** box, enter the LU Name. The LU number is system-assigned.
4. Click **OK** to exit.
5. On the **Action** menu, click **Save Configuration**.

To assign LUs and LU pools to users and groups

1. In SNA Manager, right-click the LU or LU pool, click **Assign to User**, and then select the users you want to assign.
2. Click **OK**.
3. Verify that the state of the server is active and that the connections are available (either **OnDemand** or **Active**). If the server is not active, right-click the server in SNA Manager, and then click **Start**.

To view LUs or pools assigned to users

1. In SNA Manager, double-click **Configured Users**. Then click the user or group that you want to view. The details pane shows the LUs and pools assigned to that user or group, plus other details, such as status and LU number.
2. To view local or remote APPC LUs assigned to a user or group, right-click the targeted user or group, and click **Properties** under **Configured Users**. Click the **APPC Defaults** tab. When you are done viewing, click **Cancel**.

To add 3270 LUs to a pool

1. In SNA Manager, expand **Servers**, and then expand the server you are working with.
2. Expand **Connections**, and then click the connection that contains the needed 3270 LUs.
3. In the right pane, right-click the 3270 LU that you want to assign and click **Assign Pool**.
4. In the dialog box, click the pool to which you want to assign the LU, and then click **OK**.
5. Repeat steps 2 and 3 for each LU that you want to assign to a pool.
6. On the **Action** menu, click **Save Configuration**.

Note

To verify that the LUs are in the correct pool, make sure the **Pools** node is expanded, double-click the name of the affected p ool, and view its LUs in the details pane.

Note

Both the LUs and the LU pool must be of the same type (3270).

To add a workstation and assign LUs

1. In SNA Manager, right-click **Workstations**, point to **New**, and then click **Workstation**.
2. Fill in the **Workstation Properties** dialog box.
3. Click **OK** to add the new workstation.
4. To assign LUs to the workstation, double-click **Connections**, and then double-click the connection that contains the needed LUs.
5. Right-click the LU that you want to assign, and click **Assign to Workstation**.
6. In the dialog box, click the workstation, and then click **OK**.
7. On the **Action** menu, click **Save Configuration**.

Note

All workstation users must be members of the same SNA subdomain. You can avoid having to add each user to the subdom ain by assigning the **Everyone** group to the subdomain and assigning no LUs to the group.

To remove an LU from a pool

1. In SNA Manager, click **Pools**.
2. Right-click the LU that you want to remove, and then click **Delete**.
3. In the **Confirmation** dialog box, click **Yes** to delete the LU.
4. On the **Action** menu, click **Save Configuration**.
5. On the **Action** menu, click **Refresh**.

Note

An empty pool provides no LU access.

See Also

Tasks

Creating and Configuring LUs

How to Assign LUs to Workstations

You can assign LUs to a workstation rather than a user, which makes it possible to lock LUs to a particular workstation. Assigning an LU to a workstation makes it easier for users to find and access different resources. For example, it makes it easier for a user at a workstation to use a printer located near the workstation, instead of one assigned to the user.

With this feature, you can insert a workstation object and define it by specifying parameters such as IP address, workstation name, and others. You can then assign 3270 display and printer LUs to the workstation. These LUs will be accessible to any users connecting from those workstations. The LUs do not need to be specifically assigned to the user.

For example, 200 hospital users share 50 workstations and 50 printers. Users may want to use any of the 50 workstations and 50 printers. They can log on to any of the 50 workstations and print using the printer attached to that workstation. Instead of assigning all 50 printers to 200 user accounts, the administrator can add the 50 workstations to Host Integration Server and assign a printer LU to each workstation. When a user logs on to any of the 50 workstations, the printer attached to that workstation will be available in the list of LUs.

You can also configure a workstation so that it will only allow the use of those LUs assigned to the workstation, regardless of the user who is logged on. You can assign LUs to workstations in the same manner that LUs are assigned to users.

For example, if the workstation is flagged for restricted access, a user cannot gain unauthorized access to other LUs regardless of the user ID and password that is used. This feature enhances network security.

To assign LUs to workstations

1. In the Host Integration Server console tree, right-click **Workstations**, point to **New**, and then click **Workstation**.
2. Fill in the **Workstation Properties** dialog box.
3. Click **OK** to add the new workstation.
4. To assign LUs to the workstation, click **Connections**, and then click the connection that contains the LUs.
5. Right-click the LU that you want to assign, and click **Assign to Workstation**.
6. Click the workstation, and click **OK**.
7. On the **Action** menu, click **Save Configuration**.

Note
All workstation users must be members of the same SNA subdomain. You can avoid having to add each user to the subdom ain by assigning the **Everyone** group to the subdomain and assigning no LUs to the group.

See Also

Tasks

How to Associate 3270 Printer LUs with 3270 Display LUs

How to Associate 3270 Printer LUs with 3270 Display LUs

Users who have host applications with direct relationships between display LUs and printer LUs can associate printers with the LUs.

You can associate a printer LU with a display LU. This feature is designed to support specialized host applications that have hard-coded associations between display LUs and printer LUs. When a 3270 display LU is configured to have an associated printer and then subsequently assigned to a user or group, users will see both the display LU and a special printer LU named ASSOCPRT. When users connect to the display LU, they can then open the ASSOCPRT LU, which Host Integration Server maps to the defined associated printer LU.

You can also add display LUs with associated printer LUs to LU pools. The LU pools can then be assigned to users and groups.

When multiple display LUs are assigned to users, the order in which the resources are opened is important. If all of the display LUs have associated printers, the user should alternate opening displays and ASSOCPRTs. If some of the display LUs do not have associated printers, use a naming convention so the user can determine the display LUs that have associated printers from those that do not.

To associate printer LUs to display LUs

1. In the Host Integration Server console tree, click **Pools**.
2. Right-click the display LU that you want to associate with a printer, and click **Properties**.
3. Click the **Associated Printer** tab.
4. In the **Associated Printer LU** box, select the printer that you want to associate, and then click **OK**.
5. Repeat steps 2 through 4 for each display LU that you want to associate with a printer LU.
6. On the **Action** menu, click **Save Configuration**.

Note
Printer LUs are available to any users or groups that have been assigned the corresponding display LUs.

Note
Associated printer LUs are assigned a generic placeholder name of *ASSOCPRT*.

See Also

Tasks

How to Assign LUs to Workstations

How to Configure Downstream Connections

You can configure Host Integration Server to enable downstream LUs to communicate with the host. Following are the general procedures:

1. Gather needed information about the host and downstream systems, including information about identifiers, such as addresses and exchange identification (XID), and information about Max BTU Length.

2. Configure the host connection as you would configure any other host connection. When specifying the remote end for the connection, select **Host System**.

3. Select the host connection and assign and configure one or more new downstream LUs.

4. Optionally, you can put downstream LUs into a pool to maximize their availability.

5. Configure the downstream connection. When specifying the remote end for the connection, select **Downstream**.

   The downstream connection can be an 802.2, SDLC, or X.25 connection. It cannot be a channel connection.

6. Select the downstream connection and associate the downstream LUs (from step 3) and LU pools (from step 4) with that connection.

7. If necessary, reorder the downstream LU numbers; that is, the LU numbers used by the downstream system.

   A downstream LU has two LU numbers: one recognized by the downstream system and one recognized by the host. These numbers need not match.

To put configuration changes into effect, save the configuration file and restart the server (if you have added a link service or connection).

Note
It is recommended that you manually separate the sessions of your downstream pools if you are upgrading from SNA Server 4.0 or later. If this is not done, you may experience incorrect behavior from Host Integration Server when attempting to connect downstream client computers.

After creating the downstream LUs, you must associate them with the downstream connection they will use. Downstream LUs do not stand alone and need to be mapped to the upstream LU. Associating the LUs with the upstream connection maps the correct downstream LUs to the correct upstream LUs.

Downstream LUs have two parameters:

- The LU number, which identifies the LU to the host.

- The downstream LU number, which identifies an LU or LU pool to the downstream system.

The LU number and the downstream LU number need not match, and probably will not match. The downstream LU number should match the LU number on the downstream system.

Check with the host administrator for the correct value of the LU number. It should match the LOCADDR= parameter of the LU definition in VTAM or in the NCP Gen. If the number you specify has already been assigned to an LU or an APPC LU-LU pair on the current connection, you must change the number. The range for LU numbers is 1 through 254.

Host Integration Server assigns a downstream LU number to each LU or LU pool assigned to the downstream connection. Host Integration Server assigns downstream LU numbers starting with 2 and ascending sequentially. Therefore, one way to change downstream LU numbers is to reorder the list of LUs for the downstream connection. Another way to change downstream LU numbers is to use empty LU pools to take up unwanted numbers, leaving the correct (larger) numbers for other LU pools or LUs.

Note

The LU numbers generated for an LU range are sequential. If the numbers assigned by your host administrator are not sequential, you can modify the numbers of individual LUs after they have been created.

You can move one or more downstream LUs to a different host connection, regardless of whether the LUs were created as part of a range. Although downstream LUs can be associated with multiple downstream connections, they can be assigned to only one host connection.

You can configure multiple LUs simultaneously by configuring them as a consecutively numbered range. After configuring the range of LUs, you can modify the numbering of individual LUs in the range.

To configure a downstream connection

1. In the Host Integration Server console tree, expand **Servers**, expand the server you are working with.
2. Right-click **Connections**, point to **New**, and click either **802.2**, **SDLC**, or **X.25**.
3. Fill in the **Connection Properties** dialog box, type a name for the new downstream connection, select a link service, and type a comment.
4. Under **Remote End**, select **Downstream**.
5. Under **Allowed Directions**, select an option. For SDLC downstream connections, select **Outgoing Calls**.
6. If you selected **Outgoing Calls**, under **Activation**, select either **On Server Startup** or **By Administrator**.
7. Click the other tabs to specify other connection parameters between the downstream system and Host Integration Server. Click **OK** to exit.
8. On the **Action** menu, click **Save Configuration**.
9. To put the configuration changes into effect, you must restart the server.

Note

Restarting the server disconnects all users. Try to schedule configuration changes when they least affect users.

Note

Two connections are needed for a downstream system: a downstream connection (from the downstream system to the computer running Host Integration Server) and an ordinary host connection (from the computer running SNA Server to the host).

Note

Activation does not affect incoming calls. A connection that accepts incoming calls begins to listen for these calls when the server node is started.

To assign a downstream LU to a host

1. Before you begin, verify that you have configured the host connection that the downstream LU will use.
2. Ensure that the server is inactive. If necessary, right-click the appropriate server node, and click **Stop**.
3. Double-click **SNA service**, and then double-click **Connections**.
4. Right-click the host connection to which you want to assign a downstream LU, point to **New**, and then click **Downstream LU**.
5. Enter the LU name. Click **OK** to exit.
6. On the **Action** menu, click **Save Configuration**.

Note

After you assign a new downstream LU to a host, you must also associate that LU with a downstream connection.

Note

A downstream LU pool must contain LUs from a single server. Downstream LUs cannot access pooled downstream LUs from multiple servers.

To view or modify a downstream LU

1. Right-click the LU whose properties you want to view, and click **Properties**.
2. In the **Properties** dialog box, do one of the following:
   - View the properties and click **Cancel**.
   - Modify the properties and click **OK**.
3. If you made changes, on the **Action** menu, click **Save Configuration**.
4. Stop and restart **SNA service**.

To assign a downstream LU

1. In the Host Integration Server console tree, expand **Servers**, expand the server you are working with, and then click **Connections**.
2. Right-click the host connection to which you want to assign the downstream LU, point to **New**, and click **Downstream LU**.
3. Fill in the **Connection Properties** dialog box, and click **OK**.
4. On the **Action** menu, click **Save Configuration**.

To associate a downstream LU with a downstream connection

1. Verify that the server is inactive. If necessary, right-click the service node and click **Stop**.
2. In the Host Integration Server console tree, select the downstream LUs from the host connection.

   To select several adjacent items, click the first item you want, hold down SHIFT, and click the last item.

   To select several nonadjacent items, hold down CTRL as you click the items that you want.
3. Drag the LUs from the host connection to the downstream connection.
4. On the **Action** menu, click **Save Configuration**.

Important

If a downstream system needs access to multiple LU sessions, assign one LU or LU pool for each session needed. You can ass ign a pool multiple times to the same downstream connection.

Note

Before you can associate downstream LUs with a downstream connection, you must define the host connection, downstream connection, and the LUs.

Note

Downstream LU pools must contain LUs from a single server. Downstream LUs cannot access pooled downstream LUs from multiple servers.

See Also

Tasks

Creating and Configuring LUs

How to Reorder Downstream LUs

An administrator might create a downstream LU called DOWNLU3, and then realize that two additional LUs are needed. After creating the LUs, with the names DOWNLU2 and DOWNLU4, the LUs would be associated with the listed LU numbers.

LU name	Downstream LU number
DOWNLU3	2
DOWNLU2	3
DOWNLU4	4

To change the downstream numbers so that they match the digits in the LU names, DOWNLU2 could be reordered so that it is placed before DOWNLU3.

LU name	Downstream LU number
DOWNLU2	2
DOWNLU3	3
DOWNLU4	4

To reorder the numbering of downstream LUs

1. Verify that the server is inactive. If necessary, right-click the server node, and click **Stop**.
2. In the Host Integration Server console tree, select the downstream connection with which the LUs are associated.
3. Select one or more LUs and pools that you want to move up in the list.

   To select several adjacent items, click the first item you want, hold down SHIFT, and click the last item.

   To select several nonadjacent items, hold down CTRL as you click the items that you want.
4. To move LUs up or down in the list, click the up or down arrow buttons on the Host Integration Server toolbar.
5. To put configuration changes into effect, restart the server.

To create a downstream pool

1. Right-click **Pools**, point to **New**, and then click **Downstream LU Pool**.
2. Type a pool name and, if desired, a comment. This name identifies the pool on SNA Server, not on the host or downstream system.
3. To add LUs to the new downstream pool, double-click **Connections**, and then double-click the connection that has the LUs you want to work with.
4. Verify that the **Pools** node is fully expanded, and drag the first LU into the pool.
5. To add more LUs to the pool, drag each LU from its place in the **Connections** folder to its downstream LU pool.
6. On the **Action** menu, click **Save**.

Note
Downstream LU pools must contain LUs from a single server. Downstream LUs cannot access pooled downstream LUs from multiple servers.

See Also

Tasks

Creating and Configuring LUs

How to Configure APPC LUs for TPs or 5250 Emulation

The following is an overview of the procedures to add and configure APPC LUs:

1. Assign the local APPC LU to a server.
2. Configure the local APPC LUs:
    - For an independent LU, specify the LU Alias, Network Name, and LU Name. Depending on system configuration, you may need to specify other information.
    - For a dependent LU, specify LU Alias, LU Number, and LU Name. Depending on system configuration, you may need to specify other information.
3. Create a remote APPC LU on a connection and configure the remote APPC LU:
    - For a remote LU to be used with an independent local LU, specify the LU Alias, Network Name, and LU Name. Depending on system configuration, you may need to specify other information. For communication with an AS/400 computer, make the remote LU name the same as the name of the AS/400 computer.
    - For a remote LU to be used with a dependent local LU, specify the LU Alias, Network Name, LU Name, and Uninterpreted LU Name. Depending on system configuration, you may need to specify other information.
4. Optionally, configure session security for the remote LU.
5. You must configure an appropriate mode if it has not already been done.
6. Optionally, assign a default local APPC LU and a default remote APPC LU to users and groups.
7. If you are using Common Programming Interface for Communications (CPI-C), configure one or more CPI-C symbolic destination names.
8. If you are using the display verb, you may want to change the default connection that Host Integration Server uses for the verb. To ensure that all changes take effect, restart the server.

Note
APPC LUs are used for transaction programs (TPs) or for 5250 emulation. APPC LUs used with TPs are generally for communication between TPs on different systems, not communication between TPs on a single system.

To add local and remote APPC LUs

1. In the Host Integration Server console tree, double-click **SNA service**, and then click **Local APPC LUs**.
2. On the **Action** menu, point to **New**, and click **Local LU**.
3. Configure the properties for the local LU, and then click **OK**.
4. To add the remote LU, click **Remote APPC LUs**.
5. On the **Action** menu, point to **New**, and click **Remote LU**.
6. Configure the properties for the remote LU, and click **OK**.
7. On the **Action** menu, click **Save Configuration**.

Note

APPC uses both local and remote LUs. These LUs need to be properly configured before Host Integration Server can communicate with the AS/400 computer.

Note

The option, **Member of Outgoing Local APPC LU Pool**, differs from other types of LU pools, such as the 3270, LUA, and downstream LU pools.

Note

Configuring session security for remote LUs is optional.

To configure incoming APPC sessions

1. Verify that you have configured a remote LU to represent the incoming host or that you have configured one remote LU for each host.
2. In the Host Integration Server console tree, double-click **Local APPC LUs**, and then click the local LU that you want to configure.
3. On the **Action** menu, click **Properties**.
4. Click the **Advanced** tab, specify the **Implicit Incoming Remote LU**, which is the name of the incoming host, and then click **OK**.
5. Repeat steps 2–4 for each local APPC LU that you want to configure.
6. On the **Action** menu, click **Save Configuration**.

Note

All incoming requests must specify a local LU name that is recognized by Host Integration Server, even when using an Implicit Incoming Remote LU and Implicit Incoming Mode.

To assign default APPC LUs to users or groups

1. In the Host Integration Server console tree, double-click **Configured Users**, and click the user or group that you want to configure.
2. On the **Action** menu, click **Properties**.
3. Click the **APPC Defaults** tab.
4. In the **Local APPC LU** box, select a default local APPC LU.
5. In the **Remote APPC LU** box, select a default remote APPC LU, and then click **OK**.
6. On the **Action** menu, click **Save Configuration**.

See Also

Tasks

Creating and Configuring LUs

Precedence of Accounts in Determining Default LUs

When user and group account memberships overlap, the highest-priority account that contains a default local APPC LU determines that LU for the user, and the highest-priority account that contains a default remote APPC LU determines that LU for the user. Accounts are prioritized as follows:

1. User accounts (highest priority)
2. Subdomain groups
3. Local groups
4. Well-known groups such as Everyone (lowest priority)

For example, suppose a user account (a high-priority account) called JOHND contains LOCLU1 as the default local APPC LU, but no default remote APPC LU. At the same time, suppose a local group (a low-priority account) of which JOHND is a member contains LOCLU2 as the default local APPC LU, and REMLU2 as the default remote APPC LU. For JOHND, the high-priority assignment, a default local APPC LU of LOCLU1, will be combined with the only other available assignment, a default remote APPC LU of REMLU2.

To assign or edit CPI-C information

1. In the Host Integration Server console tree, click **CPI-C Symbolic Names**.
2. On the **Action** menu, point to **New**, and click **CPI-C Symbolic Name**.
3. Fill in the **Properties** dialog box, and click **OK**.
4. On the **Action** menu, click **Save Configuration**.

Note
If you are using applications based on Common Programming Interface for Communications (CPI-C), use this procedure to c onfigure the CPI-C symbolic destination name.

To configure the default display verb

1. In SNA Manager, select the subdomain that you want to configure.
2. On the **Action** menu, click **Properties**.
3. Click the **Display Verb** tab.
4. Select the default connection for the display verb, and click **OK**.
5. On the **Action** menu, click **Save Configuration**.

Note
When the display verb does not specify a connection, Host Integration Server uses the connection that you specify in this pro cedure. If you do not specify a default display verbconnection, Host Integration Server randomly selects a connection for the verb to use.

Configuring hot backup involves setting up a system environment in which one resource can automatically fill in if another fails. In such a configuration, resources are interchangeable.

To configure hot backup for 5250 LUs

1. Configure one connection to the AS/400 computer for each computer that runs Host Integration Server and supports the 5250 protocol.
2. Create a local LU on each server, specifying the following:

- For **LU 6.2 Type**, select **Independent**.
- For **LU Alias**, specify the same string on each server.
- For **LU Name**, specify a unique string on each server. This is required to avoid device name conflicts across servers.

3. Create a remote LU on each server and give all the remote LUs the same LU name and alias, (which must match the Control Point Name of the AS/400 computer).

   Be sure that the **Supports Parallel Sessions** box is selected for these LUs.

4. Make the local and remote LUs available to users and groups, by doing one of the following:
   - For each group or user that uses 5250, select the appropriate default local APPC LU and the appropriate default remote APPC LU.
   - To keep this process simple, use or create a group that includes all 5250 users, and assign the appropriate default LUs to that group.
   - For the 5250 emulator, have users specify the local and remote LUs when configuring a session on the emulator.

Note
To achieve hot backup, Host Integration Server chooses a server to connect with the AS/400 computer based on availability. This distributes the load more evenly.

Note
To configure hot backup for 5250 users, you must configure multiple servers and pairs of LUs that can all handle sessions to the intended AS/400 computer.

Note
For better protection against failure, use multiple servers, and not multiple connections on the same server.

To configure a server with multiple connections to the same AS/400 computer

1. In the Host Integration Server console tree, double-click **Connections**.
2. Click the first connection that you want to configure.
3. On the **Action** menu, click **Properties**.
4. Click the **System Identification** tab, and fill in the properties for the local and remote node.
5. Click **OK** to exit the dialog box.
6. Repeat steps 2 through 5 for each connection to the same AS/400 computer.
7. On the **Action** menu, click **Save Configuration**.
8. After you configure a connection, you must stop and restart the server.

Important
Each connection to the same AS/400 computer must use a separate link service.

Note
This procedure makes the fully qualified network name unique for each connection.

Note

The settings that you specify in this procedure override the fully qualified network name configured in the properties for the server.

To configure a connection for dynamic definition of remote APPC LUs

1. Select the connection that you want to configure.
2. On the **Action** menu, click **Properties**.
3. On the **General** tab in the **Properties** dialog box, select **Supports Dynamic Remote APPC LU Definition**, and click **OK**.
4. On the **Action** menu, click **Save Configuration**.

See Also

Tasks

How to Configure APPC LUs for TPs or 5250 Emulation

User Management

Although Host Integration Server uses the User accounts of the Microsoft Windows Server 2003 or Windows 2000 Active Directory directory service, users must be configured to use Host Integration Server resources such as LUs.

To create users

1. In SNA Manager, expand the SNA subdomain where the users reside.
2. Right-click **Configured Users**, point to **New**, and then click **User**.
3. Select a user (or group) from the list, and then click **OK**.

To view active 3270 users

1. In SNA Manager, expand **Servers**, and then expand the server you are working with.
2. Click **Active Users**. The results pane shows the active 3270 users and provides details such as **User Name**, **Client**, and **Server**.

See Also

Tasks

How to Configure APPC LUs for TPs or 5250 Emulation

Operations

This section includes individual Users Guides for all feature sets of Host Integration Server. These guides contain the concepts, procedures, and security best practices necessary to operate your deployment of Host Integration Server.

In This Section

BizTalk Adapters

This release of Microsoft Host Integration Server includes the following four adapters:

- The **Microsoft BizTalk Adapter for Host Applications** is designed for BizTalk Server. It is based on technology in Microsoft Transaction Integrator (TI) for Windows-Initiated Processing, which enables efficient client access to existing IBM mainframe zSeries (CICS and IMS) or midrange iSeries (RPG) server programs. The TI design tools are integrated with Visual Studio and BizTalk Server solutions, enabling IT developers to be highly productive when defining the client proxy and creating the XSD schema. For BizTalk Server administrators, the existing TI Manager, Microsoft Management Console (MMC) snap-in, has been enhanced to improve supportability and support the required remote BizTalk Server solution deployment scenarios.

  For more information, see BizTalk Adapter for Host Applications.

- The **Microsoft BizTalk Adapter for Host Files** is an advanced data adapter that enables IT organizations to access and integrate information stored in host file systems, including mainframe zSeries VSAM datasets and midrange iSeries physical files. The Visual Studio 2005 design tool is used within a BizTalk Server solution to define a metadata map of the host program-described files, which is then exported as XSD for use with the BizTalk adapter. The configuration wizards are integrated into the BizTalk Server administration tools, allowing IT professionals to define dynamic send ports and static and solicit response receive ports, based on a simplified set of SQL commands (SELECT, INSERT, UPDATE, DELETE). This BizTalk adapter is based on a new Microsoft .NET Framework Data Provider for Host Files that maps SQL to non-relational host datasets and members. This makes it simple for non-programmers to read and write host file data sources.

  For more information, see BizTalk Adapter for Host Files.

- The **Microsoft BizTalk Adapter for DB2** is a relational database adapter that enables IT professionals to access vital data stored in IBM DB2 database servers on remote host computing platforms, IBM mainframe zSeries and midrange iSeries (DB2/400), and also IBM DB2 Universal Database (UDB) on open platforms. Using standard SQL commands and a configuration wizard built into BizTalk Server administration tools, IT professionals can create solutions that read and write to DB2 without any need for database programming. The new DB2 adapter, which is based on an updated Microsoft .NET Framework Data Provider for DB2, supports a broad range of functions, including dynamic send ports, and static and solicit response receive ports.

  For more information, see BizTalk Adapter for DB2.

- The **Microsoft BizTalk Adapter for WebSphere MQ (Client-Based)** uses IBM WebSphere MQ Client (Base-Client) and IBM WebSphere MQ Extended Transactional Client (Extended-Client) APIs to communicate with remote MQSeries Queue Managers. The adapter enables BizTalk Server to communicate directly with MQSeries Queue Managers deployed on non-Windows operating systems, without needing to deploy and manage WebSphere MQ Server for Windows, to efficiently exchange messages with line-of-business applications across the enterprise. When used with the Base-Client, the adapter provides non-transactional message processing, guaranteeing only the delivery of messages. It is the responsibility of the application on the receiving end to handle any duplicate messages. When used with the Extended-Client, the adapter provides transactional message processing to guarantee once-and-only-once delivery of messages.

  For more information, see BizTalk Adapter for WebSphere MQ.

In This Section

BizTalk Adapter for DB2

BizTalk Adapter for Host Files

BizTalk Adapter for Host Applications

BizTalk Adapter for WebSphere MQ

See Also

Other Resources

Operations

BizTalk Adapter for DB2

The BizTalk Adapter for DB2 is a send and receive adapter that enables BizTalk orchestrations to interact with host systems. Specifically, the adapter enables send and receive operations over TCP/IP and APPC connections to DB2 databases running on mainframe, AS/400, and UDB platforms. Based on Host Integration Server technology, the adapter uses the Data Access Library to configure DB2 connections, and the Managed Provider for DB2 to issue SQL commands and stored procedures.

The adapter serves two main functions:

- For **Send** operations (both One Way and Solicit Response), the adapter sends SQL commands and stored procedures to a DB2 instance, with the option of soliciting a response.

- For **Receive** operations (One Way only), the adapter creates an SQL command or stored procedure that polls DB2 objects and creates per-row messages, which are then submitted to the BizTalk message system.

In addition, the BizTalk Adapter for DB2 uses the standard BizTalk Adapter tracing tool as a troubleshooting mechanism.

In This Section

Installation Components

How to Create a Send Port for the DB2 Adapter

How to Create a Receive Port and a Receive Location for the Host File Adapter

How to Create a Schema for the DB2 Adapter

How to Create a BizTalk Application Using the DB2 Adapter

See Also

Other Resources

Data Access Library

Managed Provider for DB2

Installation Components

You can only install BizTalk Adapter for DB2 on a computer that is already running BizTalk Server and its prerequisites.

Required Components

The adapter installation package installs the following on your BizTalk Server computer:

- BizTalk Adapter for DB2
- Managed Provider for DB2
- Data Access Library
- Nodeless Host Integration Server server installation and configuration
- DRDA IP Resync Service (DUW)
- SNA Resync Service

Optional Components

- Distributed Units of Work for DB2
- Host Integration Server configured for APPC connections
- Microsoft Enterprise Single Sign-On

Apart from this installation, and the configuration steps outlined in the other topics in this section, no additional registration is necessary.

See Also

Other Resources

BizTalk Adapter for DB2
Data Access Library Programmer's Guide
Managed Provider for DB2 Programmer's Guide

How to Create a Send Port for the DB2 Adapter

You create a send port for the BizTalk Adapter for DB2 by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a send port

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click B**izTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Send Ports**, point to **New**, and then click **Static One-way Send Port**.

   The **Send Port Properties** dialog box appears.
4. In the **Transport Type** field, select **DB2**.
5. Click **Configure**.

   The **DB2 Transport Properties** dialog box appears.
6. Configure the following properties:

Use this	To do this
Connection String	The name of a connection string that is used to connect to the DB2 database. To configure a new or existing Connection String, click the ellipsis (**...**). This starts the Data Source Wizard. To access Help, click **Help** on the wizard pages, or open the Host Integration Server Help and look in **Technical Reference - UI Help - Data Integration Help - Data Source Wizard**.
Document Target Namespace	The target namespace that is used in the XML documents that are sent to DB2.
Response Root Element Name	The root element name that is used in the XML documents that are received from DB2. (This property may be empty for a one-way port.)
URI	Uniform resource identifier. A name to identify the send port location. Default is DB2://.

7. Click **OK** to return to the **Send Port Properties** dialog box.
8. In the **Send Handler** field, select the host instance on which the send adapter is running.
9. In the **Send Pipeline** field, select the pipeline that processes messages that are sent through this port. To configure a Send Pipeline, click "**..**".

   For more information, click **Help** on the property pages.
10. You can configure additional properties by clicking the following tabs: **Transport Advanced Options**, **Backup Transport**, **Outbound Maps**, **Filters**, **Certificate**, and **Tracking**.

    For more information click **Help** on these tabs.
11. When you are finished with configuration, click **OK** to close the **Send Port Properties** dialog box and return to the BizTalk Server Administration console tree.
12. In the **Send Ports** window, right-click the send port in the **Name** column and select **Enlist**.
13. Right-click the send port in the **Name** column and select **Start**.

See Also

Other Resources

BizTalk Adapter for DB2

How to Create a Receive Port and a Receive Location for the DB2 Adapter

You create a receive port and receive location for the BizTalk Adapter for DB2 by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a receive port and a receive location

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Receive Ports**, point to **New**, and then click **Static One-way Receive Port**.

   The **Receive Port Properties** dialog box appears.
4. Configure the properties and then click **OK**.

   For more information, click **Help**.
5. In the console tree, right-click **Receive Locations**, point to **New**, and then click **One-way Receive Location**.

   The **Select a Receive Port** dialog box appears.
6. Select the receive port you created in step 3, and then click **OK**.

   The **Receive Location Properties** window appears.
7. In the **Transport Type** field, select **DB2**, and then click **Configure**.

   The **DB2 Transport Properties** dialog box appears.
8. Configure the following properties:

Use this	To do this
Connection String	Enter the name of a connection string that will be used to connect to the DB2 database. To configure a new or existing Connection String, click the ellipsis (**...**). This starts the Data Source Wizard. To access Help, click **Help** on the wizard pages, or open the Host Integration Server Help and look in **Technical Reference - UI Help - Data Integration Help - Data Source Wizard**.
Document Root Element Name	The root element name that is used in the XML documents that are received from DB2.
Document Target Namespace	The target namespace that is used in the XML documents that are received from DB2.
SQL Command	The select or stored procedure command that is executed one time for each polling interval.

Update Command	The command that is executed after each row in the receive operation is processed. It can be either a delete statement that deletes the row from the table in the SQL command, or an update command that statically modifies one or more rows. When this option is specified, the SQL command must be a Select statement and must access a single table.
URI	A name identifying the receive port location. Default is DB2://.
Polling Interval	The number of units between polling requests. Allowed range is 1 - 65535.
Polling Unit of Measure	The unit of measure (seconds, minutes, or hours) used between polling requests. Default is seconds.
Message Batch Size	When an SQL command returns a result set to the DB2 receive adapter, the adapter submits one message to the BizTalk message engine for each row in the result set. This is referred to as a single row receive. The receive adapter can optionally return the result set in its entirety as a single message to the BizTalk message engine. Messages can also have a fixed number of rows in them (for example, one message contains 10 rows). Default is -1.

9. Click **OK** to return to the **Receive Location Properties** dialog box.
10. In the **Receive Handler** field, select the instance of the BizTalk Server host on which the receive location will run. The receive handler must be running on this host.
11. In the **Receive Pipeline** field, select the receive pipeline to use to receive messages at this receive location.
12. To configure a receive pipeline, click "**..**". For more information, click **Help** on the property pages.
13. To configure scheduling, click the **Schedule** tab.

    For more information, click **Help** on the **Schedule** tab.
14. When you are finished with configuration, click **OK** to close the **Receive Location Properties** dialog box and return to the BizTalk Server Administration console tree.
15. In the **Receive Locations** window, right-click the receive location in the **Name** column and select **Enable**.

See Also

Other Resources

BizTalk Adapter for DB2
Data Access Library
Managed Provider for DB2 Programmer's Guide
Data Access Library Programmer's Guide

How to Create a Schema for the DB2 Adapter

To create the XSD schemas for the BizTalk Adapter for DB2, you use the DB2 Schema Generation Wizard.

To generate the DB2 schema

1. Open your BizTalk Visual Studio project.
2. Right-click the project, point to **Add**, and then click **Add Generated Items**.
3. In the **Add Generated Items** dialog box, select **Add Adapter Metadata**.

   This starts the Add Adapter Wizard.
4. On the **Select Adapter** page, select **DB2**, and then click **Next**.

   This starts the DB2 Adapter Schema Generation Wizard.
5. On the **Database Information** page, create a connection string, or select an existing connection string.

   This can be Initial Catalog, Package Collection, (TCP Address and Port) or (APPC Local LU, Remote LU, and Mode), (User Name and Password), or (Integrated Security). Maximum length is 1024.
6. On the **Schema Information** page, define the default namespace, root elements, and port type to be used in the schema.

   If you select **Receive port**, only a request document root element name is needed. If you select **Send port**, both request and response document root element names are required.
7. On the **Statement Type Information** page, select the type of database command to be issued.

   If you selected receive ports on the previous page, you can choose either a SELECT SQL statement or a stored procedure. If you selected send ports on the previous page, you can choose to issue an updategram, stored procedure, or SELECT statement.
8. On the **Statement Information** page, enter the details about the DB2 database.

   Depending on the information entered on earlier pages, the following properties are available:

   a. **Receive Select Statement** If you chose to use a receive port with an SQL statement, you can either select, or browse to, the statement here.

   b. **Receive Stored Procedure** If you chose to use a receive port and issue a stored procedure, you can select a stored procedure from the current connection's catalog. You must enter values for all parameters on this page.

   c. **Send Updategram** If you chose to use a send port and updategrams, you can select an updategram operation here, and also the table and columns that will be present in the updategram.

   d. **Send Stored Procedure** If you chose to use a send port and issue a stored procedure, you can select a stored procedure from the current connection's catalog. You do not have to enter values for all parameters on this page.

   e. **Send Select Statement** If you chose to use a send port with an SQL SELECT statement, you can either select, or browse to, the statement here.

9. On the **Completing the DB2 Transport Schema Generation Wizard** page, click **Finish**.

See Also

Other Resources

BizTalk Adapter for DB2

Data Access Library Programmer's Guide

Managed Provider for DB2 Programmer's Guide

How to Create a BizTalk Application Using the DB2 Adapter

After you create the schema, ports, and receive locations for the BizTalk Adapter for DB2, you can start coding your BizTalk application.

To create a BizTalk application using the DB2 Adapter

1. Create a BizTalk project to hold your BizTalk application.
2. Use the schema that you created in How to Create a Schema for the DB2 Adapter to describe the DB2 database to the BizTalk application.
3. Use the send port that you created in How to Create a Send Port for the DB2 Adapter to send data to the DB2 database.
4. If necessary, use the receive port and location that you created in How to Create a Receive Port and a Receive Location for the DB2 Adapter to receive data from the DB2 database.
5. Add any additional orchestrations, components, or code, as necessary.
6. Test your application.
7. After you finish testing your application, create an .msi package to move your application to a staging or live server.

See Also

Other Resources

BizTalk Adapter for DB2

Data Access Library

BizTalk Adapter for Host Files

The BizTalk Adapter for Host Files is a send and receive adapter that enables BizTalk orchestrations to interact with host systems. Specifically, the adapter enables send and receive operations over TCP/IP and APPC connections to host files that run on mainframe and AS/400 platforms. Based on Host Integration Server technology, the adapter uses Data Access Library metadata assemblies to configure connections, and the Microsoft .NET Framework data provider for host files to issue SQL commands and stored procedures.

The adapter serves two main functions:

- For **Send** operations (both One Way and Solicit Response), the adapter sends SQL commands and system commands to a host file instance, with the option of soliciting a response.

- For **Receive** operations (One Way only) the adapter creates an SQL command that polls host file objects and creates per-row messages, which are then submitted to the BizTalk message system.

In addition, the BizTalk Adapter for Host Files uses the standard BizTalk Adapter tracing tool as a troubleshooting mechanism.

Note
The BizTalk Adapter for Host Files is a non-transactional adapter. This means that once an action is performed, it cannot be u ndone or rolled back.

In This Section

Installation Components

How to Create a Metadata Assembly for the Host File Adapter

How to Create a Send Port for the Host File Adapter

How to Create a Receive Port and a Receive Location for the Host File Adapter

How to Create a Schema for the Host File Adapter

How to Create a BizTalk Application for the Host File Adapter

See Also

Other Resources

Data Access Library

Managed Data Provider for Host Files

Installation Components

You can only install BizTalk Adapter for Host Files on a computer that is already running BizTalk Server and its prerequisites.

Note
To run the native 64-bit (X64) adapter, you must use a 64-bit (X64) BizTalk Server 2006 host instance.

Note
This adapter requires the SNA DDM Service. Before using the adapter, check to see that the SNA DDM Service is running by using either Services in Control Panel or the NET START SNADDM Windows command.

Note
All editions of BizTalk Server 2000, BizTalk Server 2002, and BizTalk Server 2004 can be installed on an x64 operating system but only the **BizTalk Server 2006** Enterprise and **BizTalk Server 2006** Developer editions support running a **BizTalk Server 2006** host instance using native 64-bit execution.

Required Components

The adapter installation package installs the following components on your BizTalk Server computer:

- BizTalk Adapter for Host Files
- Managed Provider for Host Files
- Data Access Library
- Nodeless Host Integration Server server installation and configuration

Optional Components

- Host Integration Server configured for APPC connections
- Microsoft Enterprise Single Sign-On

Apart from this installation, and the configuration steps that are outlined in the topics in this section, no additional registration is necessary.

See Also

Other Resources

BizTalk Adapter for Host Files
Data Access Library
Managed Data Provider for Host Files

How to Create a Metadata Assembly for the Host File Adapter

After you install the adapter, you can create a metadata assembly that describes your remote system to BizTalk Server.

To create a metadata assembly

1. Create a Host File project and application in Visual Studio.

   Part of the process of creating a Host File application in Visual Studio is describing the layout of the host file system. This process creates both a metadata assembly and a schema. The metadata assembly is a programmatic representation of the remote host file system, whereas the schema is an XML representation of the host file system. You will use the metadata assembly to describe the host file system to BizTalk Server.

   For more information about how to create a Host File application in Visual Studio, see How to Create an Application Using the Managed Data Provider for Host Files.

See Also
Other Resources
BizTalk Adapter for Host Files
Managed Data Provider for Host Files
Data Access Library

How to Create a Send Port for the Host File Adapter

You create a send port for the BizTalk Adapter for Host Files by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions on the Enterprise Single Sign-On (SSO) database.

To configure a send port

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Send Ports**, point to **New**, and then click **Static One-way Send Port** or **Solicit Response Send Port**.

   The **Send Port Properties** dialog box appears.
4. In the **Transport Type** field, select **Host File**.
5. Click **Configure**.

   The **Host File Transport Properties** dialog box appears.
6. Configure the following properties:

Use this	To do this
Connection String	The name of a connection string that is used to connect to the Host File database. To configure a new or existing connection string, click the ellipsis (**...**). This starts the Data Source Wizard. To access Help, click **Help** on the wizard pages, or open the Host Integration Server Help and look in **Technical Reference - UI Help - Data Integration Help - Data Source Wizard**.
Document Target Namespace	The target namespace that is used in the XML documents that are sent to the host.
Response Root Element Name	The root element name that is used in the XML documents that are received from the host. (This property may be empty for a one-way port.)
URI	Uniform resource identifier. A name to identify the send port location.

7. Click **OK** to return to the **Send Port Properties** dialog box.
8. In the **Send Handler** field, select the host instance on which the send adapter is running.
9. In the **Send Pipeline** field, select the pipeline that processes messages sent through this port.
10. To configure a Send Pipeline, click "**..**".

    For more information, click **Help** on the property pages.
11. You can configure additional properties by clicking the following tabs: **Transport Advanced Options**, **Backup Transport**, **Outbound Maps**, **Filters**, **Certificate**, and **Tracking**.

    For more information, click **Help** on these tabs.
12. When you are finished with configuration, click **OK** to close the **Send Port Properties** dialog box and return to the BizTalk Server Administration console tree.
13. In the **Send Ports** window, right click the send port in the **Name** column and select **Enlist**.
14. Right-click the send port in the **Name** column and select **Start**.

See Also
Tasks

How to Create a Receive Port and a Receive Location for the Host File Adapter

Other Resources

BizTalk Adapter for Host Files

Data Access Library

Managed Data Provider for Host Files

How to Create a Receive Port and a Receive Location for the Host File Adapter

You create a receive port and receive location for the BizTalk Adapter for Host Files by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a receive port and a receive location

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Receive Ports**, point to **New**, and then click **Static One-way Receive Port**.

   The **Receive Port Properties** dialog box appears.
4. Configure the properties and then click **OK**.

   For more information, click **Help**.
5. In the console tree, right-click **Receive Locations**, point to **New**, and then click **One-way Receive Location**.

   The **Select a Receive Port** dialog box appears.
6. Select the receive port you created in step 3, and then click **OK**.

   The **Receive Location Properties** window appears.
7. In the **Transport Type** field, select **HostFiles**, and then click **Configure**.

   The **HostFiles Transport Properties** dialog box appears.
8. Configure the following properties:

Use this	To do this
Connection String	Enter the name of a connection string that will be used to connect to the host database. To configure a new or existing connection string, click the ellipsis (**...**). This starts the Data Source Wizard. To access Help, click **Help** on the wizard screens, or open the Host Integration Server Help and look in **Technical Reference - UI Help - Data Integration Help - Data Source Wizard**. When configuring a receive location or send port based on the BizTalk Adapter for Host Files, the metadata definition should be created as a metadata assembly and not an HCD file. For instructions on how to create a metadata assembly, see How to Create an Application Using the Managed Data Provider for Host Files.
Document Root Element Name	The root element name that is used in the XML documents that are received from the host.
Document Target Namespace	The target namespace that is used in the XML documents that are received from the host.
SQL Command	The Select command that is executed one time for each polling interval.

Update Command	The command that is executed after each row in the receive operation is processed. It can be either a delete statement that deletes the row from the table in the SQL command, or an update command that statically modifies one or more rows. When this option is specified, the SQL command must be a Select statement and access a single table. You can specify additional properties by clicking the ellipsis (**...**) button. This opens the **Change Command** dialog box, which provides three options: • **Do nothing** clears the other two options if selected. • **Delete after read** deletes the row after the adapter has read it. • **Update** lets you type an SQL command to be updated.
URI	Uniform resource identifier. A name identifying the receive port location.
Polling Interval	The number of units between polling requests. Allowed range is 1 - 65535.
Polling Unit of Measure	The unit of measure (seconds, minutes, or hours) used between polling requests. Default is seconds.

9. Click **OK** to return to the **Receive Location Properties** dialog box.
10. In the **Receive Handler** field, select the instance of the BizTalk Host on which the receive location will run. The receive handler must be running on this host.
11. In the **Receive Pipeline** field, select the receive pipeline to use to receive messages at this receive location.

    To configure a receive pipeline, click "**..**". For more information, click **Help** on the property pages.
12. To configure scheduling, click the **Schedule** tab.

    For more information, click **Help** on the **Schedule** tab.
13. When you are finished with configuration, click **OK** to close the **Receive Location Properties** dialog box and return to the BizTalk Server Administration console tree.
14. In the **Receive Locations** window, right-click the receive location in the **Name** column and select **Enable**.

See Also

Tasks

How to Create a Send Port for the Host File Adapter

Other Resources

BizTalk Adapter for Host Files

Data Access Library

Managed Data Provider for Host Files

How to Create a Schema for the Host File Adapter

You can use the Host File Schema Generation Wizard to create the XSD schemas for the BizTalk Adapter for Host Files. After you create the schema, you are ready to continue configuration on the BizTalk Server side.

To generate the Host File schema

1. Open your BizTalk Server Visual Studio project.
2. Right-click the project, point to **Add**, and then select **Add Generated Items**.
3. In the **Add Generated Items** dialog box, select **Add Adapter Metadata**.

   This starts the Add Adapter Wizard.
4. On the **Select Adapter** page, select **Host File**, and then click **Next**.

   This starts the Host File Adapter Schema Generation Wizard.
5. On the **Database Information** page, browse to an existing connection string or create a new one.

   This can be Initial Catalog, Package Collection, (TCP Address and Port) or (APPC Local LU, Remote LU, and Mode), (User Name and Password), or (Integrated Security). Maximum length is 1024.
6. On the **Schema Information** page, define the default namespace, root elements, and port type that you want to use in the schema.

   If you select **Receive port**, only a request document root element name is needed. If you select **Send port**, request and response document root element names are required.
7. On the **Statement Type Information** page, select the type of database command to be issued.

   If you selected send ports on the previous page, you can choose to issue an updategram, stored procedure, or SELECT statement. If you selected receive ports, this step is unnecessary.
8. On the **Statement Information** page, enter the details about the host file.

   Depending on the information entered on earlier pages, the following properties will be available. If you selected send port:

   - **Send Updategram** If you chose to use a send port and updategrams, you can select the updategram operation here, and also the table and columns that will be present in the updategram.
   - **Send System Command** If you chose to use a send port and issue a stored procedure, you can select a stored procedure from the current connection's catalog. You do not have to enter values for all parameters on this page.
   - **Send Select Statement** If you chose to use a send port with an SQL Select statement, you can either select or browse to the statement here.

   If you selected receive port:

   - **Receive Select Statement** If you chose to use a receive port with an SQL statement, you can either select or browse to the statement here.
   - **Send Updategram** If you chose to use a send port and updategrams, you can select the updategram operation here, and also the table and columns that will be present in the updategram.
   - **Send Stored Procedure** If you chose to use a send port and issue a stored procedure, you can select a stored procedure from the current connection's catalog. You do not have to enter values for all parameters on this page.
   - **Send Select Statement** If you chose to use a send port with an SQL Select statement, you can either select or browse to the statement here.

9. On the **Completing the Host File Transport Schema Generation Wizard** page, click **Finish**.

See Also

Other Resources

BizTalk Adapter for Host Files

Data Access Library

Managed Data Provider for Host Files

How to Create a BizTalk Application for the Host File Adapter

After you create the schema, you can code your BizTalk application. Your application will use the metadata assembly that you created in Visual Studio, in addition to the schema and ports that you created in previous steps.

To create a BizTalk application using the Host File Adapter

1. Create a BizTalk project to hold your BizTalk application.
2. Use the schema that you created in How to Create a Schema for the Host File Adapter to describe the Host File system to the BizTalk application.
3. Use the send port that you created in How to Create a Send Port for the Host File Adapter to send data to the host file system.
4. If necessary, use the receive port and location that you created in How to Create a Receive Port and a Receive Location for the Host File Adapter
5. Add any additional orchestrations, components, or code, as necessary.
6. Test your application.
7. After you finish testing your application, create an .msi package to move your application to a staging or live server.

   When you are creating a BizTalk Server .msi package, make sure that you include the host file metadata assembly that you created in How to Create a Metadata Assembly for the Host File Adapter.

See Also

Other Resources

BizTalk Adapter for Host Files

BizTalk Adapter for Host Applications

The Microsoft BizTalk Adapter for Host Applications is based on technology in Transaction Integrator (TI) that enables enterprises to connect BizTalk Server solutions to existing IBM mainframe zSeries (CICS and IMS) or midrange iSeries (RPG) server programs. The adapter offers an intuitive Visual Studio 2005 designer, including host COBOL and RPG source code import wizards, with which to generate XSD schemas for use in BizTalk projects. The administration tools are integrated with the BizTalk Server port configuration and deployment tools. Using this adapter, IT professionals can efficiently extend existing business rules in host programs with new solutions based on BizTalk Server 2006.

Note
A receive location is not available for this adapter.

A typical scenario involves the following steps:

- The IT professional installs the adapter.
- The IT professional configures the adapter and uses Transaction Integrator to define the Remote Environment.
- The developer creates the BizTalk application in Visual Studio. This involves defining the Remote Environment, building the TI assembly and schema, deploying, testing, and debugging the application, exporting the assemblies, and building the export package for use by BizTalk Server.
- Finally, the IT professional imports the developer's application into BizTalk Server, updates any necessary information, and deploys the application.

Tracing tools are available to debug your deployed application and improve its performance.

Some knowledge of Transaction Integrator is necessary. For an overview of TI basics, see the Transaction Integrator User's Guide.

In This Section

Installation Components

The following software must be installed on the appropriate computers.

Development Computer

The following software must be installed on the computer where the developer will create the BizTalk application:

- Windows Server 2003
- Visual Studio 2005
- SQL Server 2005
- BizTalk Server 2006
- BizTalk Adapter for Host Applications (or Host Integration Server)

Deployment Computer

The following software must be installed on the computer where the IT professional will deploy the BizTalk application:

- Windows Server 2003
- SQL Server 2005
- BizTalk Server 2006
- BizTalk Adapter for Host Applications

Installing the Adapter

If you installed Host Integration Server, the BizTalk Adapter for Host Applications was installed by default. If it was not, or if you have uninstalled it, rerun the Host Integration Server Setup program. When you reach the **Select Components** screen, select **Adapters** and then select **BizTalk Adapter for Host Applications**.

See Also

Other Resources

BizTalk Adapter for Host Applications

How to Configure SSO for the Host Application Adapter

When using the BizTalk Adapter for Host Applications with the BizTalk Server Service Account, it is necessary to create and map a Single Sign-On Affiliate Application.

To create and map the Affiliate Application

1. In the **Enterprise Single Sign-On Management Console**, create an SSO Affiliate Application. For more information, see How to Create an Affiliate Application.
2. Create a mapping for the Host Instance Account. For more information, see How to Create User Mappings.
3. In **TI Manager**, right-click the **Host Instance Account**, and then click **Properties**.
4. Click the **Security** tab.
5. In the **Affiliate Application** field, choose the SSO mapping you just created, and click **OK**.
6. In the BizTalk Server Administration console, right-click the **Send Port** for this adapter, and then click **Properties**.
7. On the **Properties** page, confirm that **SSO Affiliate Application** is set to **<Use RE settings>**.
8. Confirm that **Allow application to override selected authentication** is not selected.
9. Click **OK**.

How to Create a Send Port for the Host Application Adapter

You create a send port for the BizTalk Adapter for Host Applications by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a send port

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Send Ports**, point to **New**, and then click **Solicit Response Send Port**.

   The **Send Port Properties** dialog box appears.
4. In the **Transport Type** field, select **HostApps**.
5. Click **Configure**.

   The **HostApps Transport Properties** dialog box appears.
6. Configure the following properties:

Use this	To do this
Host Type	Defines the type of host that the adapter instance will interact with. The Host Type is a three part name: "TI" <TI Programming model / RE Style> <RE Override name> • The first part of the name is always TI. • The second part is the TI RE Style. • The third part is the optional RE Override Name. You can change the host type or RE Override by clicking the ellipsis (**...**) to start the **Select Host Type and RE Override** dialog box.
TI Remote Environment Type	Displays a list of all available Remote Environment class types. Selecting <Any host type> will allow message processing with no restrictions.
RE Override	Enables the adapter administrator to change the Remote Environment destination at run time. This is useful if, in addition to the central production RE, a test or backup RE is required. If selected, this will apply to all messages that flow through the specific adapter instance. This option is only available when the TI Remote Environment Type is set to a valid RE Class, and is not available if the TI Remote Environment Type is set to <Any host type>.

Implicit Persistent Connections	Enables the adapter to use TI persistent connections in the case where a batch of messages is delivered to the adapter for processing. The adapter evaluates the messages in the batch to determine the viability of using a persistent connection. For example, persistent connections could be used in the following scenarios: • When TRM Link, ELM Link, CICS Link, IMS Connect or DPC are used, and all messages are associated with the same TI object. • When TRM User Data, ELM User Data, or CICS User Data is used, and all messages are associated with the same TI object and method. If it is selected, the adapter will sort the messages into logical groups to facilitate the use of persistent connections.
Use Transactions	Allows a remote environment to be used with or without transactions through multiple instances of this adapter.

7. Click **OK** to return to the **Send Port Properties** dialog box.
8. In the **Send Handler** field, select the host instance on which the send adapter is running.
9. In the **Send Pipeline** field, select the pipeline that processes messages that are sent through this port.
10. To configure a Send Pipeline, click "**..**".

    For more information, click **Help** on the property pages.
11. You can configure additional properties by clicking the following tabs: **Transport Advanced Options**, **Backup Transport**, **Outbound Maps**, **Filters**, **Certificate**, and **Tracking**.

    For more information, click **Help** on these tabs.
12. When you are finished with configuration, click **OK** to close the **Send Port Properties** dialog box and return to the BizTalk Server Administration console tree.
13. In the **Send Ports** window, right-click the send port in the **Name** column and select **Enlist**.
14. Right-click the send port in the **Name** column and select **Start**.

See Also

Other Resources

BizTalk Adapter for Host Applications

Creating an Application for the BizTalk Adapter for Host Applications

The following list describes the steps that you must follow to create an application that uses the BizTalk Adapter for Host Applications:

1. Set your mainframe environment and communications protocols to run your application.

2. Create a Visual Studio solution to contain all the necessary BizTalk and Host Integration Server projects for your application.

3. Create a Transaction Integrator (TI) project that will hold the interface definition of the host application.

   If necessary, you may modify the generated XML file to pass client context information.

4. Create a BizTalk project that will hold your BizTalk application, using the created interface definition and associated schema.

5. Create your BizTalk application, using the ports and settings that were defined earlier.

6. Associate the interface definition with the mainframe environment using TI Manager.

7. Deploy the schema.

8. Build the BizTalk Server deployment MSI package.

After you create the export package, you can move the solution to a staging or production server and import the package.

For more information, see Application Integration Programmer's Guide.

In This Section

Mainframe Setup

How to Create a Visual Studio Solution

How to Create a Transaction Integrator Project and Interface Definition

How to use a Client Context with the BizTalk Adapter for Host Applications

How to Create a BizTalk Project

Creating a BizTalk Application

How to Associate the Interface Definition with the Mainframe Environment

How to Export the Schema

How to Create a BizTalk Server Export Package

Mainframe Setup

The primary purpose of the BizTalk Adapter for Host Applications is to connect to an application that is running on a remote mainframe. Therefore, the first step in writing your BizTalk application is to confirm that the host mainframe is, in fact, running the specified application, and that you can access the mainframe from your system.

How to create, load, and troubleshoot a remote host application is beyond the scope of this topic. You should work with your mainframe developer to create and load an application onto the specified mainframe. You can also work with a mainframe system developer to access the mainframe from your system.

In addition, the SDK for the BizTalk Adapter for Host Applications includes a detailed readme on how to create and load a sample mainframe application. For more information, see [*installation directory*]\Microsoft Host Integration Server\SDK\Samples\AppInt\BizTalkHostApplications\. This directory contains instructions on how to set up a mainframe sample application, in addition to several variations on how to set different network protocols for the sample.

See Also

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

How to Create a Visual Studio Solution

After you create, debug, and confirm your access to the host application, you can start to create the application for the BizTalk Adapter for Host Applications. The first task you must perform is to create a Visual Studio solution. This solution will contain the BizTalk Server and Host Integration Server projects that will, in turn, contain the necessary components for your application.

To create a Visual Studio solution

1. In Visual Studio, click **File**, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, in the Project types pane, click **Other Project Types**, and then click **Visual Studio Solutions**.
3. In the Templates pane, click **Blank Solution**.
4. Enter the name of your project in the **Name** field, and then click **OK**.

See Also

Tasks

How to Create a Transaction Integrator Project and Interface Definition

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

Mainframe Setup

How to Create a Transaction Integrator Project and Interface Definition

When you are creating an application for the BizTalk Adapter for Host Applications, you first create a Visual Studio solution to contain your projects, and then you create a Transaction Integrator (TI) project. Using the TI project, you will create an interface definition of the host application. Later, you will use the interface definition, together with an associated .xml schema, to describe the interface to your BizTalk application.

To create a TI project in Visual Studio

1. In your Visual Studio solution, click **File**, point to **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Project types** pane, click **Host Integration Projects**.
3. In the **Templates** pane, click **Transaction Integrator Project**.
4. Enter the name of your TI project in the **Name** field, enter the location to save the project in the **Location** field, and then click **OK**.

To add a .NET client library to the TI project

1. In Solution Explorer, right-click the name of your TI project, point to **Add**, and then click **Add .NET Client Library**.
2. In the **Add New Item** dialog box, confirm that .**NET Client Library** is selected in the **Templates** pane.
3. Type the name of the client library in the **Name** field, and then click **OK**.
4. On the **.NET Client Library Wizard Welcome** page, click **Next**.
5. On the **Library** page, in the **Type Restrictions** field, select **BAHA**.
6. Enter the relevant name, version number, and description, and then click **Next**.
7. On the **Remote Environment** page, enter the relevant information that describes the remote environment, click **Next**, and then click **Create**.

   If you do not know this information, you should speak to your mainframe systems developer.

To construct the interface definition

1. In Host Integration Server Designer, right-click the node of the .NET client library, point to **Import**, and then click **Host Definition**.
2. Use the Import COBOL Wizard to import a host definition file.

   By importing the host definition file into the .NET client library, HIS Designer also creates an XML schema data (XSD) file. This XSD file contains the interface definition you will use later in your BizTalk application. For more information about using the Import COBOL Wizard, see Import COBOL Wizard.
3. You may view the XSD file by clicking the **XSD Definition** tab in HIS Designer.
4. Click **File**, and then click **Save All**.

   The XSD file should be saved in the standard location for the project. Note that you will need this location in the next step, when you add the XSD file to your BizTalk project. For more information, see How to Create a BizTalk Project.

See Also
Tasks
How to Create a Visual Studio Solution
How to Create a BizTalk Project
Concepts
Creating an Application for the BizTalk Adapter for Host Applications

How to use a Client Context with the BizTalk Adapter for Host Applications

For Host Integration Server, the client context is a set of values that describe the context in which the client application is running in the Windows environment. By modifying the **ClientContext** keyword in the XML file associated with your BizTalk project, you can pass the client context information on to the remote application.

To use client context keywords in a BizTalk Adapter for Host Applications application

1. Generate an instance of the XML document that describes your assembly.

   For more information, see How to Create a Transaction Integrator Project and Interface Definition

2. In the XML document, edit the **TIClientContext** keyword.

   The **TIClientContext** keyword will appear in the XML document in the following format:

   ```
   <ns0:TIClientContext TIContextKeyword="TIContextKeyword_0" TIContextValue="TIContextValue_1" />
   ```

   Where "TIContextKeyword_0" and TIContextValue_1" represent the keyword and value pair, respectively.

   For the BizTalkAdapter for Host Applications, **TIClientContext** accepts USERID, PASSWORD, RecvTimeOut, and SendTimeOut as valid keywords.

   Note that you do not have to remove **TIClientContext** elements from your document if you are not using **ClientContext**.

3. When finished, save the XML file and exit.

Example

The following example describes an XML document generated by the Transaction Integrator Designer.

```
<ns0:DPC_WGB__GetBalance__WGBANK__Request xmlns:ns0="http://microsoft.com/HostApplications/
TI/WIP">
  <ns0:WGBANKInDocument>
   <ns0:ACCNUM>1234</ns0:ACCNUM>
   <ns0:name>Kim Akers</ns0:name>
  </ns0:WGBANKInDocument>
  <ns0:TIClientContext TIContextKeyword=" RecvTimeOut " TIContextValue="15" />
<ns0:TIClientContext TIContextKeyword="TIContextKeyword_0"TIContextValue="TIContextValue_1"
/>
</ns0:DPC_WGB__GetBalance__WGBANK__Request>
```

See Also

Other Resources

COMTIContext Keywords

How to Create a BizTalk Project

After you create an XSD file for your application, you create a BizTalk project to contain the application. After you create the BizTalk project, you can add the XSD file as a standard item, and then use it to create your BizTalk application.

To create a BizTalk project for an application that uses the BizTalk Adapter for Host Applications

1. In Visual Studio, click **File**, point to **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Project Types** pane, click **BizTalk Projects**.
3. In the **Templates** pane, click **Empty BizTalk Server Project**.
4. Type the name of the project in the **Name** field, type the location of the project in the **Location** field, and then click **OK**.

To add the .xsd file to the BizTalk project

1. In Solution Explorer, right-click the name of your BizTalk project, select **Add**, and then click **Existing Item**.
2. In the **Add Existing Item** dialog box, navigate to the location where you saved the .xsd file in the previous step, click the .xsd file, and then click **Add**.

See Also

Tasks

How to Create a Transaction Integrator Project and Interface Definition

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

Creating a BizTalk Application

Creating a BizTalk Application

After you have created a BizTalk project and added the Transaction Integrator (TI) schema, you can start to create your BizTalk application. For more information, see the BizTalk Server documentation.

See Also

Tasks

How to Create a BizTalk Project

How to Associate the Interface Definition with the Mainframe Environment

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

How to Associate the Interface Definition with the Mainframe Environment

After you create a BizTalk project for your application that uses the BizTalk Adapter for Host Applications, and then add the .xsd file to the project, you must identify your target mainframe, and then associate the interface definition with the mainframe environment.

To configure the remote environment using TI Manager

1. Start TI Manager.
2. Double-click **Transaction Integrator** in the console tree.
3. Right-click **Remote Environments**, point to **New**, and then click **Remote Environment**.
4. Use the New Remote Environment Wizard to describe the environment on your target mainframe.

   For more information about how to use the Remote Environment Wizard, see Creating and Managing Remote Environments Using TI Manager

To administer the TI .NET assembly metadata using TI Manager

1. Start TI Manager.
2. Double-click **Transaction Integrator** in the console tree.
3. In the **Windows-Initiated Processing** node, right-click **Objects**, point to **New**, and then click **Object**.
4. Use the Object Wizard to associate the TI assembly that you created earlier with the remote environment of your target mainframe.

   For more information about how to use the Object Wizard, see Object Wizard (for WIP).

See Also

Tasks

How to Export the Schema

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

Creating a BizTalk Application

How to Export the Schema

After you associate the interface with the mainframe environment, you must deploy the schema. Deploying the schema exposes an artifact from Visual Studio to BizTalk Server.

To deploy a schema

1. In Visual Studio, in Solution Explorer, right-click the name of your BizTalk project, and then click **Deploy**.

See Also

Tasks

How to Associate the Interface Definition with the Mainframe Environment

How to Create a BizTalk Server Export Package

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

How to Create a BizTalk Server Export Package

After you finish testing your BizTalk application in your development environment, you can put the associated dependencies into an export package. After you create the export package, you can then move your BizTalk application to a staging or live server.

Note
The process for creating an export package for a BizTalk application that uses the BizTalk Adapter for Host Applications is identical to creating any other export package that uses an adapter.

To create an export package for an application that uses the BizTalk Adapter for Host Applications

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Server 2006 Administration**, expand the BizTalk group, and then expand **Applications**.
3. Right-click the application that you want to export, point to **Export**, and then click **MSI file**.
4. On the **Welcome to the Export MSI File Wizard** page, click **Next**.
5. On the **Select Resources** page, select the artifacts to export into the .msi file, and then click **Next**.
6. If you are prompted, on the **Specify IIS Hosts** page, type the server name of the computer hosting the virtual directory that you want to include, and then click **Next**.

   You will be prompted to specify the server only if the virtual directory has not been previously added to the BizTalk Management database, such as when it was added to the application or was imported in an application.
7. On the **Dependencies** page, review the dependencies for the application, and then click **Next**.
8. On the **Destination** page, in **Destination application name**, type the application name.
9. In **MSI file to generate**, type the full path for the .msi file, and then click **Export**.
10. On the **Summary** page, note the location of the log file for this operation, and then click **Finish**.

See Also

Tasks

How to Export the Schema

How to Deploy the BizTalk Server Application

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

How to Deploy the BizTalk Server Application

After a developer has created and exported an application for the BizTalk Adapter for Host Applications, an IT professional can import it, update the connection properties, and deploy the application.

Before you import the application, you can find more information by completing the Importing Exported Packages tutorial. After you complete these procedures, you can perform any application-specific deployment tasks as necessary.

To import the BizTalk Server application

1. In the BizTalk Server Administration console, expand the **BizTalk Server 2006 Administration** node.
2. Expand the **BizTalk Group** node.
3. Right-click the **Applications** node, click **Import**, and then click **MSI file**.

   This starts the Import MSI Wizard.
4. Follow the instructions in the wizard. If you need help, you can click **Help** on each wizard page.

To update the connection properties

1. In the TI Manager Console, expand the **Remote Environments** folder.
2. Right-click the **Remote Environment** for this application, and then click **Properties**.

   - If you have a TCP/IP connection, select the **TCP/IP** tab, and either confirm or enter the appropriate information in the **IP/DNS Address** and **Port** fields.

   - If you have an SNA connection, select the **LU6.2** tab, and either confirm or enter the appropriate information in the **Local LU Alias**, **Remote LU Alias**, and **Mode Name** fields.

     You may have to obtain this information from your Mainframe or System Administrator.

See Also
Concepts
Creating an Application for the BizTalk Adapter for Host Applications
Other Resources
BizTalk Adapter for Host Applications

BizTalk Adapter for WebSphere MQ

The Client-Based BizTalk Adapter for WebSphere MQ (MQSC Adapter) is a connectivity solution that enables you to use BizTalk Server in an enterprise with WebSphere MQ as the chosen messaging standard.

Previously, once-and-only-once delivery of messages between BizTalk Server and WebSphere MQ applications was provided by the Server-Based BizTalk Adapter for WebSphere MQ, which requires MQSeries Server on Windows as an intermediate server between BizTalk Server and non-Windows Queue Managers. To enable once-and-only-once delivery of messages, BizTalk Server and the adapter require WebSphere MQ to participate in a distributed transaction using MSDTC (Microsoft Distributed Transaction Coordinator). MSDTC support has been available only with the Server version of WebSphere MQ on Windows.

With BizTalk Server 2006, transactional messaging (once-and-only-once delivery) is also available through the MQSC Adapter. This is made possible by the MQSC Adapter working together with the WebSphere MQ Extended Transactional Client (MQ Extended-Client). Like the MQSeries Server, the MQ Extended Client supports distributed transactions using Microsoft Distributed Transaction Coordinator (MSDTC) on Windows. Therefore, the adapter can guarantee once-and-only-once delivery of messages by ensuring that both BizTalk Server and MQ Extended-Client participate in a distributed transaction.

When receiving messages from MQSeries and submitting them to BizTalk Server, the adapter starts an MSDTC transaction and performs an MQGet with SYNCPOINT so that MQSeries participates in the transaction. The adapter passes this same transaction context to BizTalk Server so that BizTalk Server participates in the same transaction when the adapter submits the message to it. After the message has been submitted, the adapter commits the transaction. When sending messages from BizTalk Server to MQSeries, the adapter starts the transaction and performs an MQPut operation with the SYNCPOINT option. BizTalk Server uses this same transaction to remove the message from the BizTalk Server MessageBox database, after which the adapter commits the transaction.

You can also configure the MQSC Adapter to support non-transactional messaging when integrating with MQSeries Queues. For this, the MQSC Adapter uses the WebSphere MQ Base-Client. In this case, the adapter only guarantees that no messages are lost. Duplication of messages can occur under failure conditions. Therefore, you should use this configuration option only if the application that is consuming the message from BizTalk Server or MQSeries Queues can handle duplication of messages. To prevent loss of messages, the MQSC adapter first does an MQGET with a browse lock by setting the MQGMO_BROWSE_FIRST and MQGMO_LOCK options. The adapter then submits the message to BizTalk Server. If the submitted message to BizTalk Server is successful, the adapter does a destructive MQGet with MQGMO_MSG_UNDER_CURSOR option. If a failure occurs while submitting the message to BizTalk Server, the adapter does an MQGet with MQGMO_UNLOCK so that additional operations can be performed on the message.

Both the Server-Based BizTalk Adapter for WebSphere MQ and the Client-Based BizTalk Adapter for WebSphere MQ offer their own advantages. The Client-Based Adapter was not designed to replace the Server-Based Adapter. Instead, it provides an additional option for integration between BizTalk Server and WebSphere MQ.

The following table compares the client-based MQSC adapter with the server-based MQSC adapter.

Feature	**Server-Based BizTalk Adapter for WebSphere MQ (MQS)**	**Non-Transactional Client-Based BizTalk Adapter for WebSphere MQ (MQSC)**	**Transactional Client-Based BizTalk Adapter for WebSphere MQ (MQSC)**
WebSphere MQ Dependency	Requires WebSphere MQ Server on Windows to communicate with WebSphere MQ Queue Managers on non-Windows Systems. This can be on BizTalk Server or on a remote server running Windows.	Requires WebSphere MQ Client to be installed on BizTalk Server to communicate directly to WebSphere MQ Queue Managers on remote systems.	Requires WebSphere MQ Extended Transactional Client to be installed on BizTalk Server to communicate directly to WebSphere MQ Queue Managers on remote systems.
Receive capability	Yes	Yes	Yes
Static send ports	Yes	Yes	Yes
Dynamic send ports	Yes	Yes	Yes

Polling queues on receive	Yes, with static MQGMO Wait Interval for three seconds.	Yes, with configurable MQGMO Wait Interval.	Yes, with configurable MQGMO Wait Interval.
Supports transactional or non-transactional scenarios	Only transactional scenarios are supported. Non-transactional configuration is available for test/debug mode, but not supported in production.	Non-transactional only.	Transactional only.
Guarantees once-and-only-once delivery of messages	Yes	No, in failure conditions, duplicate messages can occur either in BizTalk Server or in MQSeries queues. Application is responsible for handling duplicate messages.	Yes
Prevents loss of messages	Yes	Yes	Yes
Performance and scalability characteristics	Provides highest performance; better suited to handle heavy message loads.	Compared to server-based adapter, performance is low because of overhead built in to prevent loss of messages.	Performance is higher than non-transactional adapter, but lower than server-based adapter.
Receive-side conversion	When performing MQGET, MQGMO CONVERT option is specified when configured.	When performing MQGET, MQGMO CONVERT option is specified when configured.	When performing MQGET, MQGMO CONVERT option is specified when configured.
Send-side conversion	Can be configured to convert to code page of MQSeries Server on Windows.	Not applicable	Not applicable
Access to MQSeries headers from Orchestrations and Pipeline Components	Yes	Yes	Yes
Segmentation using Queue Manager capabilities	Yes	Yes	Yes
Security between BizTalk Server and MQSeries Server	COM+ application (MQSAgent) on MQSeries Server on Windows uses COM+ roles to allow users who can access it. On the wire, data is encrypted using Packet Privacy. MQSeries Server on Windows to remote MQSeries Server on non-Windows system can be configured to use SSL.	Configure Secure Sockets Layer (SSL) between MQSeries Client and Server	Configure SSL between MQSeries Client and Server
Dynamically receives from queue using solicit-response send port based on certain match options	Yes	No	No

MQSeries Channel configuration on BizTalk Server	No	Yes, uses Server Connection Channel.	Yes, uses Server-Connection Channel. To use SSL, Client Channel Definition file must be used.

In This Section

MQSC Adapter Features

How to Install the MQSC Adapter

How to Add the MQSC Adapter to a BizTalk Server Installation

How to Configure a Send Port for the MQSC Adapter

How to Configure a Receive Port and a Receive Location for the MQSC Adapter

How to Configure a Client Channel Definition File

How to Configure the MQSC Adapter for Transactional Messaging

How to Configure SSL for the MQSC Adapter

MQSC Adapter Schema

MQSC Adapter Features

The Client-Based BizTalk Adapter for WebSphere MQ (MQSC Adapter) is designed for BizTalk Server and works with all the BizTalk Server components and tools. With the adapter, you can do the following:

- Communicate with remote Queue Managers that are running on Windows or other operating systems directly from BizTalk Server.
- Send messages to MQSeries Remote Queue Definitions, Local queues, Transmission queues, and Alias queues from BizTalk Server.
- Receive messages from MQSeries Transmission queues, Local queues, and Alias queues.
- Support clustered MQSeries Queue Managers.
- Support clustered BizTalk servers.
- Poll MQSeries Queues with wait interval.
- Configure Static, Dynamic, Solicit-Response send ports, and Static Receive Locations for this adapter.
- Map context properties to header properties for both transmitting and receiving messages. Gain easy programmatic access to MQSeries header properties (including MQMD, MQXQH, MQCIH, and MQIIH) through BizTalk context properties.
- Receive messages in batches from MQSeries queues.

The MQSC Adapter also does the following:

- Enables correlation with either BizTalk Server or MQSeries using the correlation identifier.
- Provides ordered delivery of messages.
- Provides Secure Sockets Layer (SSL) support for secure communication with remote MQSeries Queue Managers.

The MQSC Adapter can co-exist with the Server-Based Microsoft BizTalk Adapter for WebSphere MQ (MQSeries Adapter).

See Also

Concepts

How to Install the MQSC Adapter

Other Resources

BizTalk Adapter for WebSphere MQ

How to Install the MQSC Adapter

The BizTalk Adapter for WebSphere MQ (MQSC Adapter) is installed as part of the BizTalk Adapters for Host Systems installation process.

To use the adapter with the IBM WebSphere MQ Client (Base-Client), you must also have the following software installed:

- Microsoft BizTalk Server 2006.
- Microsoft .NET Framework 2.0.
- IBM WebSphere MQ Client 5.3 with CSD10, **or** IBM Websphere MQ Client 6.0 with Fix Pack 6.0.1.1.

To use the MQSC adapter for Extended-Client support, you must also have the following software installed:

- IBM WebSphere MQ Client 5.3 with CSD10 and IBM WebSphere MQ Extended Transactional Client, **or** IBM WebSphere Client 6.0 with Fix Pack 6.0.1.1.

To install the MQSC Adapter

1. Run the Host Integration Server Setup.
2. On the component installation screen, expand **BizTalk Adapters**.
3. Select **BizTalk Adapter for WebSphere MQ (Client-Based)**.

As with other adapters, you can uninstall through the **Add or Remove Programs** utility in Control Panel.

Note
When you are using Extended-Client and configuring the adapter send port, receive port, and receive location, refer to How to Configure the MQSC Adapter for Transactional Messaging for more information

Refer to IBM documentation for more information about how to obtain and install WebSphere MQ Extended Transactional Client or WebSphere MQ Client. For information about how to obtain necessary fixes from IBM, refer to the README file

Important
Only x86 (32-bit) Windows operating systems that are supported by BizTalk Server 2006 are supported by the MQSC Adapter. WebSphere MQ on Windows is not supported on 64-bit Windows operating systems. This means that the MQSC Adapter is not supported on either X64 (64-bit) Windows or a 32-bit BizTalk Host Instance on x64.

See Also

Tasks

How to Add the MQSC Adapter to a BizTalk Server Installation

Other Resources

BizTalk Adapter for WebSphere MQ

How to Add the MQSC Adapter to a BizTalk Server Installation

When you install the BizTalk Adapter for WebSphere MQ (MQSC Adapter), the BizTalk Adapters for Host Systems installation process adds the adapter to your BizTalk Server installation by default. Following installation, if for any reason the adapter is missing from the BizTalk Server 2006 Administration Console (for example, if it was deleted during testing), you can add it manually by using the following procedure.

To add the MQSC adapter

1. In the Console Root of the BizTalk Server 2006 Administration Console, expand the **BizTalk Server 2006 Administration** node.
2. Expand the **BizTalk Group** node.
3. Expand the **Platform Settings** node.
4. Right-click **Adapters**.
5. Click **New**.

   The Adapter Properties window is displayed.
6. In the **Name** field, type a name for the adapter.
7. In the **Adapter** field, select **MQSC** in the list.
8. In the **Description** box, type any text that is useful to you. (This is an optional step.)
9. Click **OK**.

Note
Adding the adapter requires that you restart the host instance.

See Also
Concepts
How to Install the MQSC Adapter
Other Resources
BizTalk Adapter for WebSphere MQ

How to Configure a Send Port for the MQSC Adapter

You configure a send port for the BizTalk Adapter for WebSphere MQ by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a send port

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a send port.
3. Right-click **Send Ports**, point to **New**, and then click **Static One-way Send Port**.
4. In the **Send Port Properties** window, select **mqsc** in the **Transport Type** list.
5. Click **Configure**.
6. In the adapter's **Transport Properties** window, configure the send port properties (refer to the tables at the end of this procedure).

Note
The following properties are required to configure a send port: **Channel Name** (This is a case-sensitive property.) **Connection Name** **Transport Type** **Queue** (This is a case-sensitive property.) **Queue Manager** (This is a case-sensitive property.)

   If you do not specify a **Channel Name** property, you must provide a client channel definition file to enable the WebSphere MQ Client installed on the BizTalk Server computer to communicate with remote queue managers. You must also provide a client channel definition file if you configure Secure Sockets Layer (SSL) to work with transactional messaging. For more information, see How to Configure a Client Channel Definition File.
7. When you have finished configuring the properties, click **OK**.
8. In the **Send Port Properties** window, in the **Send handler** list, select the host instance on which the send adapter is running.
9. In the **Send pipeline** list, select the pipeline that processes the messages sent through this port.
10. Click **OK**.
11. In the **Send Ports** window, right-click the send port in the **Name** column and select **Enlist**.
12. Right-click the send port in the **Name** column and select **Start**.

    In the **Advanced** section of the **Transport Properties** window, you can set the following send port properties.

Use this	To do this
SSO Affiliate	Sets the Single Sign-On (SSO) affiliate application. The user ID and password from SSO are used for the MQMD_UserIdentifier, and the MQIIH_Authenticator (or MQCIH_Authenticator) property respectively. This assumes that a valid SSO ticket exists. Default: Blank

Transaction Supported	When this property is set to **Yes**, the MQSC adapter works in conjunction with the WebSphere MQ Extended Transactional Client (Extended-Client) on the BizTalk Server computer to prevent loss of messages and to guarantee once-and-only-once delivery of messages. When it is set to **No**, duplication of messages may occur. In this case, the adapter uses the non-transactional WebSphere MQ Client (Base-Client) for integration with MQSeries. Default: No

In the **Channel Definition** section of the **Transport Properties** window, you can set the following properties.

Use this	To do this
Channel Name	Name of the channel defined on the MQSeries Server computer that the client communicates with. This must be a 'Server Connection' Channel type. Note that this is a case-sensitive property.
Connection Name	Name of the MQSeries Server that contains the Queue Manager and Queues that the MQSC Adapter sends messages to. For the TCP transport type, the format to be specified is SERVERNAME(PORT). The port number is equivalent to the port number defined in the Listener associated with the Queue Manager. The server name can also be specified as an IP address. For LU6.2, specify the LU Name or LU Pool Name configured in Host Integration Server.
Heart Beat	Number of seconds between checks to verify that the client/server connection is working. Default: 300
Password	Password that can be used by the MCA when attempting to initiate a secure LU 6.2 session with a remote MCA. The initial value of this optional property is null.
SSL Cipher Specification	Defines a single CipherSpec for an SSL connection that will be used by the end-point configured in the adapter. Both ends of a WebSphere MQ SSL channel definition must include the attribute, and the value specified here should match the name specified on the server end of the channel. The value is a string with a maximum length of 32 characters. Required only when SSL is configured for communication between the MQSeries Client and remote Queue Managers.
SSL Peer Name	Used to check the distinguished name (also known as DN) of the certificate from the peer queue manager or client at the other end of a WebSphere MQ channel. If the distinguished name that is received from the peer does not match this value, the channel does not start. Required only when SSL is configured for communication between the MQSeries Client and Queue Managers.
Transport Type	TCP and LU6.2 are supported. Default: TCP
User Id	MCA user identifier that is used by MQSeries MCA for authorization to access MQSeries resources. The initial value is null. This is an optional property. When this attribute is blank, the MCA uses its default user identifier.

In the **MQSeries** section of the **Transport Properties** window, you can set the following properties.

Use this	To do this

Segmentation Allowed	Set to **Yes** to tell MQSeries Queue Manager to create segmented messages when submitting large messages to MQSeries Queues. Default: **No**

In the **Queue Definition** section of the Transport Properties window, you can set the following properties.

Use this	To do this
Queue	MQSeries queue to which the adapter will send messages. Remote Queue Definitions, Local Queues, Alias Queues, and Transmission Queues are supported. Note that this is a case-sensitive property.
Queue Manager	Name of MQSeries Queue Manager that contains the queues to which the adapter will send messages. Clustered Queue Managers are supported. Note that this is a case-sensitive property.

13. Click **OK**.

See Also

Tasks

How to Configure a Receive Port and a Receive Location for the MQSC Adapter

Concepts

How to Configure a Client Channel Definition File

Other Resources

BizTalk Adapter for WebSphere MQ

How to Configure a Receive Port and a Receive Location for the MQSC Adapter

You configure a receive port and receive location for the BizTalk Adapter for WebSphere MQ by using the BizTalk Server Administration console. You must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, you must have appropriate permissions in the Single Sign-On (SSO) database.

To configure a receive port and receive location

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.
2. In the console tree, expand **BizTalk Group**, expand **Applications**, and then select the application for which you want to create a receive port.
3. Right-click **Receive Ports**, point to **New**, and then click **One-way Receive Port**.
4. In the **Receive Port Properties** window, configure properties for the port, and then click **OK**.
5. In the console tree, right-click **Receive Locations**, point to **New**, and then click **One-way Receive Location.**
6. In the **Select a Receive Port** window, click the receive port that you created in the previous step, and then click **OK**.
7. In the **Receive Location Properties** window, select the MQSC adapter as the Transport Type, and then click **Configure**.
8. In the adapter's **Transport Properties** window, configure the receive location's properties (refer to the tables at the end of this procedure).

Note
The following properties are required to configure a receive location: **Channel Name** (This is a case-sensitive property.) **Connection Name** **Transport Type** **Queue** (This is a case-sensitive property.) **Queue Manager** (This is a case-sensitive property.) If you do not specify a **Channel Name** property, you must provide a client channel definition file to enable the WebSp here MQ Client installed on the BizTalk Server computer to communicate with remote queue managers. You must also provide a client channel definition file if you configure Secure Sockets Layer (SSL) to work with transactional messagin g. For more information, see How to Configure a Client Channel Definition File.

9. When you have finished configuring the properties, click **OK**.
10. In the **Receive Location Properties** window, in the **Receive handler** list, select the instance of the BizTalk Server host on which the receive location will run.

    The receive handler must be running on this host.
11. In the **Receive pipeline** list, select the receive pipeline to use to receive messages at this receive location.
12. Click **OK**.
13. In the **Receive Locations** window, right-click the Receive Location in the **Name** column and select **Enable**.

    Receive Location Properties

    In the **Advanced** section of the **Transport Properties** window, you can set the following properties.

Use this	**To do this**

Data Offset for Headers	The adapter uses values from the MQSeries headers (the MQMD, MQXQH, MQIIH, and MQCIH structures) and populates corresponding values in the BizTalk Server context properties. By default, the adapter removes these MQSeries properties from the message body. Set to **No** to retain the properties in the message body. Default: **Yes**
Event Log Error Threshold	The maximum number of the same error to be logged for certain error conditions. The adapter continues operating and, if the adapter recovers, it logs the event in the Application log. Default: 10
Ordered	Set to **Yes** to maintain the order of the messages as they are received from the MQSeries queue and submitted to the BizTalk Server MessageBox. For the send side, the adapter sends the message to the queue in the same order that it receives it from the message box. Set to **No** to not maintain message order. For send-side ordering, if you are not using Orchestration, you must enable Ordered Delivery in the Transport Advanced Options of the send port configuration. For receive-side ordering, if you are using Orchestration, you must also set the **Ordered Delivery** property to **True** in your orchestration for the receive location. Ordered delivery can decrease performance; unless you require ordered delivery, it is not recommended. Default: **No**
Stop on error	Set to **Yes** to stop processing if there is an error. This option ends the transaction and disables the receive location if there is an error. Default: **No**
Suspend As Non Resumable	Set to **Yes** to move a message to the suspended queue when there is an error and indicate if it is resumable or not. Enabling this value does not preserve ordered delivery when there is an error, but does allow the receive location to continue receiving messages. Default: **No**
Transaction Supported	When set to **Yes**, the MQSC adapter works together with the WebSphere MQ Extended Transactional Client (Extended-Client) on the BizTalk Server computer to prevent loss of messages and to guarantee once-and-only-once delivery of messages. When set to **No**, duplication of messages may occur. In this case, the adapter uses the non-transactional WebSphere MQ Client (Base-Client) for integration with MQSeries. Default: **No**
Wait Interval	When MQGet is performed to retrieve messages from MQSeries Queue, MQGMO option for Wait Interval can be set. If there are no messages in the queue, the adapter waits for the specified time (in seconds) before closing the client request. As soon as messages arrive on the queue, the adapter starts retrieving the messages. Default: 3

In the **Channel Definition** section of the Transport Properties window, you can set the following properties.

Use this	To do this

Channel Name	Name of the channel defined on the MQSeries Server computer that the client communicates with. This must be a 'Server Connection' Channel type. Note that this is a case-sensitive property.
Connection Name	Name of the MQSeries Server that contains the Queue Manager and Queues that the MQSC Adapter receives messages from. For the TCP transport type, the format to specify is SERVERNAME(PORT). Port number is equivalent to the port number defined in the Listener associated with the Queue Manager. The server name can also be specified as an IP address. For LU6.2, specify the LU Name or LU Pool Name configured in Host Integration Server.
Heart Beat	Number of seconds between checks to verify that the client/server connection is working. Default: 300
Password	Password that can be used by the MCA when trying to initiate a secure LU 6.2 session with a remote MCA. The initial value of this optional property is null.
SSL Cipher Specification	Defines a single CipherSpec for an SSL connection that will be used by the end-point configured in the adapter. Both ends of a WebSphere MQ SSL channel definition must include the attribute, and the value specified here should match the name that is specified on the server end of the channel. The value is a string with a maximum length of 32 characters. Required only when SSL is configured for communication between the MQSeries Client and remote Queue Managers.
SSL Peer Name	Used to check the distinguished name (also known as DN) of the certificate from the peer queue manager or client at the other end of a WebSphere MQ channel. If the distinguished name that is received from the peer does not match this value, the channel does not start. Required only when SSL is configured for communication between the MQSeries Client and Queue Managers.
Transport Type	TCP and LU6.2 are supported. Default: TCP
User Id	MCA user identifier that is used by MQSeries MCA for authorization to access MQSeries resources. The initial value is null. This is an optional property. When this attribute is blank, the MCA uses its default user identifier.

In the **MQSeries** section of the Transport Properties window, you can set the following properties.

Use this	To do this
Character Set	Character set to which the message should be converted when messages are received from the MQSeries Que ue. If this property is set to a value other than None, the adapter sets the MQGMO CONVERT option when perf orming an MQGet. **None**: Do not convert. **UCS-2 and UTF-16**: Convert to these character sets. MQSeries does not distinguish between them. UTF-8: Convert to the UTF-8 character set. Default: None

Segmentation Allowed	Set MQSeries to assemble segmented messages or to get the message as is. Use No Action to read messages from the MQSeries queue without enabling segmentation. Use Complete Message to have MQSeries assemble segmented messages before passing them on to the adapter. Default: **No Action**

In the **Performance** section of the Transport Properties window, you can set the following properties.

Use this	To do this
Maximum Batch Size	Maximum size of a batch of messages in KB. This property and **Maximum Messages in Batch** work together so that the limit is whichever value the adapter reaches first. Default: 100
Maximum Messages in Batch	Maximum number of messages from 1 to 10,000 in a batch. This property and **Maximum Batch Size** work together so that the limit is whichever value the adapter reaches first. Default: 10
Threads	Number of threads used per receive location. Default: 2

In the **Queue Definition** section of the Transport Properties window, you can set the properties listed in the following table.

Use this	To do this
Queue	MQSeries queue from which the adapter will receive (MQGet) messages. Transmission Queues, Local Queues, Alias Queues are supported. Note that this is a case-sensitive property.
Queue Manager	Name of MQSeries Queue Manager that contains the Queues from which the adapter will retrieve messages. Clustered Queue Managers are supported. Note that this is a case-sensitive property.

14. Click **OK**.

See Also
Tasks
How to Configure a Send Port for the MQSC Adapter
Concepts
How to Configure a Client Channel Definition File
Other Resources
BizTalk Adapter for WebSphere MQ

How to Configure a Client Channel Definition File

To specify a channel definition, you must provide a client channel definition file for WebSphere MQ Client components to use if the following are true:

- When configuring send and receive ports, you did not specify a **Channel Name** property.
- When configuring send and receive ports, you set the **Transaction Supported** property to Yes, and you configured Secure Sockets Layer (SSL) for client/server communication for WebSphere MQ.

To configure a client channel definition file

1. On your WebSphere MQ Server computer, create the client channel definition file.

   For information about how to create a client channel definition file, refer to IBM WebSphere MQ product documentation.

   After the file is defined, a binary format .TAB file is created. By default, this file is named AMQCLCHL.TAB, and it is typically located in the <*WebSphere MQ Server installation folder*>\qmgrs\<QueueManagerName>\@ipcc folder.

2. Move the AMQCLCHL.TAB file to the WebSphere MQ client computer on which BizTalk Server is installed, and define the MQCHLLIB and MQCHLTAB environment variables on this computer.

   - For MQCHLLIB, specify the folder that contains the AMQCLCHL.TAB file. By default, it is the WebSphere MQ client installation folder.
   - For MQCHLTAB, specify the name of the .TAB file (by default AMQCLCHL.TAB).

When you are setting up SSL using a client channel definition file, the key repository environment variable (MQSSLKEYR) must also be set on the WebSphere MQ client computer on which BizTalk Server is installed. For MQSSLKEYR, specify the path of the key repository for the client.

See Also

Other Resources

BizTalk Adapter for WebSphere MQ

How to Configure the MQSC Adapter for Transactional Messaging

After you install the IBM WebSphere MQ Extended Transactional Client on your BizTalk Server computer, the following additional configuration steps are necessary before you can implement transactional messaging with the BizTalk Adapter for WebSphere MQ.

- In the WebSphere MQ Server environment, give your Network Service account appropriate permissions, as described in the IBM Technote article 1223479. For security reasons, it is strongly recommended that you use the "Security Exit" so that you do not have to add "Network Service" account into the MQM group.

- On your BizTalk Server computer, add the MQSeries XA dll to the MSDTC registry. To the registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\XADLL**, add the string value **amqmtsxatmc.dll** in the **Name** column and add its path to the **Data** column. Provide the path in the form **<WebSphere MQ Client installation folder>\bin\amqmtsxatmc.dll**; for example, **C:\Program Files\IBM\WebSphere MQ\bin\amqmtsxatmc.dll**.

- On your BizTalk Server computer, if you are using WebSphere MQ 5.3, give your Network Service account read/write access to the @SYSTEM folder, contained in <WebSphere MQ Client installation folder>\qmgrs\@SYSTEM. (You do not have to do this if you are using WebSphere MQ 6.0.)

- Make sure that MSDTC is enabled on the computer on which BizTalk Server is installed and that security is configured as described in the following procedure:

To enable MSDTC and configure security

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Component Services**.
2. In the Console Root of the Component Services Console, expand **Component Services**.
3. Expand **Computers**.
4. Right-click **My Computer**, and then click **Start MSDTC**.
5. Right-click **My Computer**, and then click **Properties**.
6. Click the **MSDTC** tab.
7. Under **Transaction Configuration**, click **Security Configuration**.
8. Under **Security Settings**, select the **Network DTC Access** and **Enable XA Transactions** check boxes.
9. Click **OK**.

See Also
Other Resources
BizTalk Adapter for WebSphere MQ

How to Configure SSL for the MQSC Adapter

The following procedures are designed to help you with configuring a Windows MQSeries Client to run with Secure Sockets Layer (SSL)-enabled MQSeries Server channels. The procedures describe configuration for one-way (Server) authentication.

Configuration is performed in the following steps:

- Set up the Queue Manager/Client to work without SSL.
- Add SSL to the configuration.
- Configure the MQSeries Client-Based Adapter properties for SSL.

Note
For more information, refer to IBM WebSphere MQ documentation. If you already have MQSeries client/server SSL working, you can go directly to the procedure for configuring the SSL properties in the adapter.

The following procedures assume that you are setting up a new Queue Manager. However, you can also apply these steps to existing Queue Managers.

To set up the Queue Manager/Client to work without SSL

1. Create a Queue Manager named QM1, and define a listener on the required port.
2. Define a SVRCONN channel TO.QM1.
3. Define a CLNTCONN channel TO.QM1.
4. Supply the name on the SVRCONN channel to which it will connect (TO.QM1), the transport type, the IP address/DNS name of the server, and the port number.
5. Define a local queue on the target Queue Manager named TESTQUEUE, which can be used for testing the client connections.
6. Copy the AMQCLCHL.TAB file from the server onto the client computer. (This file can be found in /var/mqm/qmgrs/<QueueManagerName>/@IPCC on most UNIX installations and /Program Files/<Websphere MQ Server installation folder>/qmgrs/<QueueManagerName>/@IPCC on most Windows installations.)
7. On the client computer, set the following environment variables:
   - MQCHLLIB=C:\sslclient\ssl\ (where MQCHLLIB is set to the path of the client channel table).
   - MQCHLTAB=AMQCLCHL.TAB (where MQCHLTAB is set to the name of the client channel table).

Note
There are defaults for these environment variables if you want to use them. See the WebSphere MQ Client manual for more information.

8. Test that the client connection works by running amqsputc.exe on your BizTalk Server computer: **amqsputc.exe TESTQUEUE**.

To add SSL to the configuration

1. Add the certificate to the Queue Manager's store (using Internet Explorer/the MQSeries user interface or amqmcert on Windows, or gsk6ikm or gsk6cmd on UNIX).
2. Alter the SVRCONN channel so the SSLCIPH is set (for example, to NULL_MD5) and set SSLCAUTH to OPTIONAL.

Note

SSLCAUTH is required for two-way authentication (client/server).

3. Alter the CLNTCONN channel so the SSLCIPH is set to the same as the SVRCONN channel (for example, to NULL_MD5).
4. Copy the new AMQCLCHL.TAB file from the server onto the client computer; the changes made for SSL can be picked up.
5. On the Windows client computer, ensure that the CA certificates are in the system key store (you can do this from Internet Explorer) and if they are not, import them into it (again, using Internet Explorer).
6. Export the following environment variable to specify the location and name of the client key store: **set MQSSLKEYR=C:\sslclient\ssl\key**.

Note

The key store **must** have the file name extension .sto and the environment variable **must not** specify it.

7. When you have the required CA certificates in the system store, you can set up a client key store.
   a. List the certificates in the system CA store: **amqmcert -l -k ca** and note the number(s) of the required CA certificate(s)

   b. Add the certificates to the client store: **amqmcert -a (certificate_number)**, where (certificate_number) is the number of each required certificate.

8. Test that the SSL Client connections work by using the amqsputc sample program and the test queue that you set up previously.

Note

You do not actually have to import CA certificates into the Windows system store before; for example, you can import t he certificates to the client certificate store straight from a file. See the IBM MQSeries System Admin guide for informat ion about amqmcert.

When the MQSeries Client-to-MQSeries Queue Manager SSL is working, the adapter can be configured on both receive locations and send ports to use SSL. The property values that were used in the test must be specified in the adapter configuration. The following adapter properties are relevant to both send port and receive locations:

SSL Cipher Specification defines a single CipherSpec for an SSL connection that will be used by the endpoint configured in the adapter. Both ends of a WebSphere MQ SSL channel definition must include the attribute, and the value specified here should match the name that was specified on the server end of the channel. The value is a string with a maximum length of 32 characters.

SSL Peer Name is used to check the distinguished name (also known as DN) of the certificate from the peer queue manager or client at the other end of a WebSphere MQ channel. If the distinguished name received from the peer does not match this value, the channel does not start.

See Also
Other Resources
BizTalk Adapter for WebSphere MQ

MQSC Adapter Schema

The BizTalk Adapter for WebSphere MQ uses the same property schema assembly (MQSeries.dll) that is available with the server-based MQSeries Adapter. Because the server-based MQSeries Adapter is available with BizTalk Server 2006, this assembly should be already deployed in the BizTalk Management Database.

In addition, an extension schema assembly is available with the MQSC Adapter. This property schema assembly is called MQSeriesEx.dll, and it contains properties that are valid only to the Client-Based MQSeries Adapter. The assembly is deployed into the BizTalk Management Database as part of the adapter installation.

For information about these context properties in both property schema assemblies (MQSeries.dll and MQSeriesEx.dll), refer to the Programming Guide section.

If these assemblies are not deployed, you can deploy them by using the btsdeploy command-line utility.

At the command prompt, type the following:

btsdeploy DEPLOY assembly=<*Path to MQSeries.dll or MQSeriesEx.dll***>**

Both assemblies can be found in the MQSC Adapter installation location.

See Also

Other Resources

BizTalk Adapter for WebSphere MQ

Data Integration User's Guide

The Data Integration User's Guide contains information about the Data Access Tool (DAT). The DAT provides a single mechanism to help configure network, security, and database information. The DAT also helps you create packages on the DB2 system.

In This Section

Data Access Tool

Data Access

Data Access Tool

The Data Access Tool (DAT) is a new feature in Host Integration Server designed to streamline the process of creating and managing connections to data sources on host environments. Whereas this used to be divided into multiple feature sets and programs, the DAT provides a single mechanism to guide you through configuring network, security, and database information, and creating packages on the DB2 system.

The DAT consists of two items:

The **Data Source Browser** displays all data sources in familiar Microsoft Management Console (MMC) format for configuration and management.

The **Data Source Wizard** guides you step by step through the configuration process. The wizard dynamically adapts to both DB2 and File System data sources, and displays the appropriate screens.

Note
The Data Access Tool and DB2 data providers create DB2 packages for use by all DB2 users, by setting the execute privilege o n the DB2 packages to PUBLIC. It is recommended that you revoke execute privileges to PUBLIC on these packages, and then grant execute to selected DB2 users or groups.

In This Section

How to Edit a Configuration

Editing the configuration brings up the editor for the specific data source. In other words, OLE DB data sources trigger the Universal Data Links page, and ODBC data sources trigger the Data Access Tool Wizard.

To edit a configuration

1. In the **Data Source Browser** window, on the **Action** menu, click **Edit Data Source**.

   The **Data Source** property page appears.

2. When you are finished editing, click **OK**.

See Also
Other Resources
Data Access Tool

How to Display an Initialization String

This option displays the native initialization string, for example, an OLE DB connection string or an ODBC connection string, in the output window. This string can be copied from the output window and pasted into another text-based window.

To display an initialization string

1. In the **Data Source Browser** window, on the **Action** menu, click **Display Connection String**.

   The connection string appears for viewing or copying.

See Also
Other Resources
Data Access Tool

How to Test a Connection

You can test the connection to the data source and display information about it, such as the host platform and version. In a connection between an AS/400 and a DB2, a typical result of this operation might look like the following:

```
Successfully connected to data source 'My_SOURCE'.
Server class: DB2/400
Server version: 05.01.0000
```

Certain operations may require the user to enter a user name and password. If this is the case, the **Password** dialog box appears.

To test a connection

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Test Connection**.

See Also
Other Resources
Data Access Tool

How to Create Packages

The create package command creates Host Integration Server packages on a DB2 system. As the process runs, status messages are displayed in the results pane. As seen on an AS/400, the result of this operation is:

```
Connected to data source 'MY_SOURCE_IP'.
AUTOCOMMITTED package has been created.
READ COMMITTED package has been created.
READ UNCOMMITTED package has been created.
SERIALIZABLE package has been created.
REPEATABLE READ package has been created.
The package creation process has completed successfully.
```

To create packages

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Create Packages**.

   The **Create Packages** dialog box appears.

See Also

Other Resources

Data Access Tool

How to Run a Sample Query

You can perform a sample query against the host data source. The query retrieves a list of tables from the system catalogs using the default schema property configured in the data source. The data is displayed in the results pane as two tabs: an **Output** tab and a **Grid** tab.

To run a sample query

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Sample Query**.

See Also

Other Resources

Data Access Tool

How to Convert Data Sources

Converting data sources allows you to convert a DB2 data source from one provider type to another. Only acceptable provider type conversion is displayed. For example, converting from DB2 OLE DB to DB2 OLE DB is not enabled.

To convert a data source

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Convert To**, and then select a provider from the list.

See Also

Other Resources

Data Access Tool

How to Add a Table

This command displays the **Table** dialog box. You can also access this dialog box by selecting a table and clicking **Properties** from the context-sensitive menu.

To add a table

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Add Table**.

   The **Table** property page appears.
3. Enter the appropriate information in the **Table** property page.

   For more information about these properties, click **Help**.
4. When you are finished, click **OK**.

See Also
Other Resources
Data Access Tool

How to Create a New Data Source or Data Description

Clicking **New Data Source** initiates the Data Source Wizard. Clicking **New Data Description** creates an empty data description file and adds it to the tree view.

To create a new data source or data description

1. In the **Data Source Browser** window, click the **File** menu.
2. Click **New**, and then click **Data Source** or **Data Description**.

See Also
Other Resources
Data Access Tool

How to Open a File

Use this command to choose a .udl or .dsn file through a standard **File Open** dialog box. The command will open the data source using the Data Source Wizard.

To open a file

1. In the **Data Source Browser** window, click the **File** menu.
2. Click **Open Data Source**.

   The **File Open** dialog box appears.
3. Locate the appropriate data source and then click **Open**.

See Also
Other Resources
Data Access Tool

How to Import a File

This command enables you to import a file that contains the information in a DB2 Connect exported file (TCP/IP only). The values in the file are parsed and displayed in the Data Source Wizard.

To import a file

1. In the **Data Source Browser** window, click the **File** menu.
2. Click **Import**, and then click the appropriate file.

See Also

Other Resources

Data Access Tool

Other Tasks

The following are additional user interface components in the Data Access Tool that you can use to perform tasks in Host Integration Server.

Edit menu

- Use the **Edit** menu to **Undo**, **Cut**, **Copy**, or **Paste** strings, and to **Delete** or **Remove** data sources.

View menu

- Use the **View** menu to **Refresh** the browser or view the Data Access Tool **Options** dialog box.

Help menu

- Use the **Help** menu to access Host Integration Server Help.

Toolbar buttons

- The toolbar buttons located at the top of the browser window offer single-click access to menu commands. To display a function, rest the pointer over a button.

See Also
Other Resources
Data Access Tool

Data Access

Data services included with Host Integration Server enable you to interact with vital data sources, including host file systems and DB2 databases.These services are available for both the hierarchical and peer environments.

In This Section

OLE DB Provider for AS/400 and VSAM

To use Microsoft OLE DB Provider for AS/400 and VSAM with an OLE DB consumer application, you must either create a universal data link (.udl) file and call this from your application, or call the provider using a connection string that includes the provider name. Microsoft Data Access Components (MDAC) includes Microsoft Data Link, a generic method for managing and loading connections to OLE DB data sources. Microsoft Data Link also supports finding and storing connections to OLE DB data sources.

For additional information about using .udl files or connection strings, see How to Create a Connection String for a .udl File or Configuring Data Descriptions in the Programmer's Guide.

Note
The SNAOLEDB provider runtime and Data Access Tool both support a full path to the HCD files. To maximize security, you s hould store HCD files in a secure local folder or share that only the developer and the runtime application can access.

See Also

Tasks

How to Browse OLE DB Data Sources

Reference

Configuring a Data Source for OLE DB Provider for AS/400 and VSAM

Concepts

How to Add or Configure a Data Link for Windows

Data Description for AS/400 and VSAM

With the Microsoft OLE DB Provider for AS/400 and VSAM, you can directly access record-level data in mainframe VSAM, Partitioned Data Sets (PDS), and midrange OS/400 files from an application that uses OLE. The OLE database (OLE DB) is a standard set of interfaces that provides heterogeneous access to disparate sources of information located anywhere — file systems, e-mail folders, and databases. The OLE DB Provider for AS/400 and VSAM combines the universal data access of OLE DB with the record-level input/output (RLIO) protocol of the IBM Distributed Data Management (DDM) architecture.

In addition to describing tables that are used by the Microsoft OLE DB Provider for AS/400 and VSAM, you can describe tables for the File Transfer utility. To indicate that the File Transfer Utility should use a given table, click the **Properties** page for the specific table, and then select the **Use Table for File Transfer** check box. Leaving this box clear indicates that the table is to be used by the Microsoft OLE DB Provider for AS/400 and VSAM.

In addition to the sample applications provided on the Host Integration Server CD-ROM, many of the sample programs that ship as part of the Microsoft Data Access Components (MDAC) SDK can be used with the OLE DB Provider for AS/400 and VSAM. To use the OLE DB Provider for AS/400 and VSAM, you must specify **SNAOLEDB** for the provider name.

For more information about using the Host Data Description utility, see Host CCSID and Data Description in the OLE DB Providers Programmer's Guides.

See Also

Concepts

OLE DB Provider for AS/400 and VSAM

Other Resources

Data Access

OLE DB Provider for DB2

The Microsoft OLE DB Provider for DB2 contains the following features:

- Interactive and scriptable Setup program
- SNA Trace Utility and Trace Viewer
- TCP/IP network connection
- Execution of dynamic SQL commands (DDL and DML), including CALL statement for stored procedures
- Customized Data Link property dialog boxes for creating and modifying file-persisted OLE DB data link files.

To use the Microsoft OLE DB Provider for DB2 with an OLE DB consumer application, you must either create a universal data link (.udl) file and call it from your application, or call the provider using a connection string that includes the provider name. Microsoft Data Access Components (MDAC) version 2.0 and later includes Microsoft Data Link, a generic method for managing and loading connections to OLE DB data sources. Microsoft Data Link also supports finding and storing connections to OLE DB data sources.

For more information about using .udl files or connection strings, see the OLE DB Provider for DB2 Programmer's Guide in the **OLE DB Providers Programmer's Guides**

How to Browse OLE DB Data Sources

By default, data links are created in the \Program files\Common files\System\Ole db\Data links folder. A shortcut is provided in the Host Integration Server program group.

To browse OLE DB data sources

1. Click **Start**, point to **Programs**, and then point to **Host Integration Server**.
2. Point to **Data Integration**, and then click **OLE DB Data Source Browser**.

   The list of data links appears.

See Also

Concepts

OLE DB Provider for DB2

OLE DB Provider for AS/400 and VSAM

How to Create Packages for DB2

Microsoft OLE DB Provider for DB2 and Microsoft ODBC Driver for DB2 are implemented as IBM Distributed Relational Database Architecture (DRDA) Application Requesters. These features use DB2 packages to issue dynamic SQL statements and call DB2 stored procedures. The provider and driver create packages dynamically in the location to which the user points by using the Package Collection attribute in the data source definition.

To start the Create Package utility

1. Click **Start**, point to **Programs**, and then point to **Host Integration Server**.
2. Click **Data Integration**, and then select **Packages for DB2**.

See Also

Concepts

OLE DB Provider for DB2
OLE DB Provider for AS/400 and VSAM
Create Package Utility

Create Package Utility

Microsoft OLE DB Provider for DB2 and Microsoft ODBC Driver for DB2 are implemented as IBM Distributed Relational Database Architecture (DRDA) application requesters. These features use DB2 packages to issue dynamic SQL statements and call DB2 stored procedures. The provider and driver create packages dynamically in the location to which the user points using the Package Collection attribute in the data source definition.

By default, the provider automatically creates one package in the target collection, if one does not exist, at the time the user issues the first SQL statement (or calls SQL catalog to fetch schema information). The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the User ID value that is configured in the data source. The package is created for use by SQL statements issued under the same isolation level that is associated with the transaction isolation level property and parameter. If no transaction isolation level is specified, the default for mainframe DB2 is CS and for other platforms the default is NC.

Multiuser Environments

A problem can occur in multiuser environments. For example, if User A specifies a Package Collection value that represents a DB2 collection used by multiple users, but User A does not have authority to GRANT execute rights to the packages to other users (for example, PUBLIC), the package is created for use only by User A. This means that User B might be unable to access the required package. The solution is for an administrative user who has package administrative rights (for example, PACKADM authority in DB2 for OS/390) to create a set of packages for use by all users.

Host Integration Server offers the Create Packages for DB2 utility, which an administrator can use to create packages. This utility can be run using a privileged User ID to create packages in collections accessed by multiple users. The utilities create the following sets of packages and grant EXECUTE privilege to PUBLIC for all:

- AUTOCOMMIT package (MSNC001)
- READ_UNCOMMITTED package (MSUR001)
- REPEATABLE_READ package, (MSRR001)
- READ_COMMITTED package, (MSCS001)
- SERIALIZABLE or REPEATABLE_READ package (MSRS001)

Once created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, DB2/400 QSYS.SYSPACKAGE, and DB2 UDB SYSIBM.SYSPACKAGE catalog tables.

Note
When you are upgrading to Host Integration Server from Microsoft SNA Server 4.0 Service Pack 2 or Service Pack 3, you mu st re-create any existing packages by running the CrtPkg utility.

See Also

Tasks

How to Create Packages for DB2

Concepts

Support for Isolation Levels Using the ODBC Driver for DB2

OLE DB Provider for DB2

Other Resources

Support for Isolation Levels Using the OLE DB Provider for DB2

ODBC Driver for DB2

Microsoft ODBC Driver for DB2 enables access over SNA LU 6.2 and TCP/IP networks to remote DB2 databases. This driver is implemented as an IBM Distributed Relational Database Architecture (DRDA) application requester that can connect to most DRDA-compliant DB2 systems, including MVS, OS/390, OS/400, AIX RS/6000, and Microsoft Windows.

You can use the driver interactively or from an application program to issue SQL statements and execute DB2 stored procedures. From Microsoft Office Excel, users can import DB2 tables into worksheets and use Excel graphing tools to analyze the data. From Microsoft Office Access, users can import from and export to DB2. Using Microsoft Internet Information Services (IIS), developers can publish DB2-stored information to users through a Web browser interface.

See Also

Tasks

How to Add an ODBC Data Source

Other Resources

Data Access

How to Add an ODBC Data Source

Follow these steps to add an ODBC data source.

To add an ODBC data source

1. Click **Start**, point to **Settings**, and then click **Control Panel**.
2. Double-click **ODBC Data Sources**.
3. Click **Add**.
4. Click **ODBC Driver for DB2**, and then click **Finish**.

For additional information about specific ODBC Driver for DB2 parameters, see
Configuring a Data Source for the ODBC Driver for DB2.

Note
You can also display the ODBC Data Sources configuration tool from the shortcut located in the Host Integration Server program group.

See Also
Other Resources
Data Access
Data Integration User's Guide

How to Create Packages for DB2

Microsoft OLE DB Provider for DB2 and Microsoft ODBC Driver for DB2 are implemented as IBM Distributed Relational Database Architecture (DRDA) Application Requesters. These features use DB2 packages to issue dynamic SQL statements and call DB2 stored procedures. The provider and driver create packages dynamically in the location to which the user points using the Package Collection attribute in the data source definition.

To launch the Create Package utility

1. Click **Start**, point to **Programs**, and then point to **Host Integration Server**.
2. Point to **Data Integration**, and then click **Packages for DB2**.

See Also

Tasks

How to Add an ODBC Data Source

Concepts

Create Package Utility

Create Package Utility

Microsoft OLE DB Provider for DB2 and Microsoft ODBC Driver for DB2 are implemented as IBM Distributed Relational Database Architecture (DRDA) Application Requesters. These features use DB2 packages to issue dynamic SQL statements and call DB2 stored procedures. The provider and driver create packages dynamically in the location to which the user points using the Package Collection attribute in the data source definition.

By default, the provider automatically creates one package in the target collection, if one does not exist, at the time the user issues the first SQL statement (or calls SQL catalog to fetch schema information). The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the User ID value that is configured in the data source. The package is created for use by SQL statements issued under the same isolation level associated with the transaction isolation level property and parameter. If no transaction isolation level is specified, the default for mainframe DB2 is CS, and the default for other platforms is NC.

Multiuser Environments

A problem can occur in multiuser environments. For example, if User A specifies a Package Collection value that represents a DB2 collection that is used by multiple users, but User A does not have authority to GRANT execute rights to the packages to other users (for example, PUBLIC), the package is created for use only by User A. This means that User B might be unable to access the required package. The solution is for an administrative user who has package administrative rights (for example, PACKADM authority in DB2 for OS/390) to create a set of packages for use by all users.

Host Integration Server offers the Create Packages for DB2 utility, which an administrator can use to create packages. This utility can be run using a privileged User ID to create packages in collections accessed by multiple users. The utilities create the following sets of packages and grant EXECUTE privilege to PUBLIC for all:

- AUTOCOMMIT package (MSNC001)
- READ_UNCOMMITTED package (MSUR001)
- REPEATABLE_READ package, (MSRR001)
- READ_COMMITTED package, (MSCS001)
- SERIALIZABLE or REPEATABLE_READ package (MSRS001)

For a table that maps the DB2 isolation levels to the ANSI isolation levels, see the following topics:

- Support for Isolation Levels Using the OLE DB Provider for DB2
- Support for Isolation Levels Using the ODBC Driver for DB2

Once created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, DB2/400 QSYS.SYSPACKAGE, and DB2 UDB SYSIBM.SYSPACKAGE catalog tables.

Note
When you are upgrading to Host Integration Server from Microsoft SNA Server 4.0 Service Pack 2 or Service Pack 3, you must re-create any existing packages by running the CrtPkg utility.

See Also

Tasks

How to Create Packages for DB2

How to Add an ODBC Data Source

File Transfer

Host File Transfer gives a user the ability to move a file between the local computer and a host system.

In This Section

Host File Transfer

Host File Transfer

Host File Transfer gives a user the ability to move a file between the local computer and a host system. Host Integration Server provides this service by using a single ActiveX control. This extends the ability for the client application to perform file transfer operations from many client development environments.

For an example of the correct usage of the File Transfer ActiveX control, see the following SDK sample:

SDK/Samples/FileTransfer/TestConnectVB/TestConnect.exe

Samples in this directory can also be used as fully functional utilities.

See Also
Concepts
Data Description

Data Description

With the Microsoft OLE DB Provider for AS/400 and VSAM, you can directly access record-level data in mainframe VSAM, Partitioned Data Sets (PDS), and midrange OS/400 files from an application that uses OLE. The OLE Database (OLE DB) is a standard set of interfaces that provides heterogeneous access to disparate sources of information located anywhere — file systems, e-mail folders, and databases. The OLE DB Provider for AS/400 and VSAM combines the universal data access of OLE DB with the record-level input/output (RLIO) protocol of IBM's Distributed Data Management (DDM) architecture.

In addition to describing tables that are used by the Microsoft OLE DB Provider for AS/400 and VSAM, you can describe tables for the File Transfer utility. To indicate that a given table is to be used by the File Transfer utility, select the **Properties** page for the table and select the **Use Table for File Transfer** check box. Leaving this box cleared indicates that the table is to be used by the Microsoft OLE DB Provider for AS/400 and VSAM.

In addition to the sample applications provided on the Host Integration Server CD-ROM, many of the sample programs that ship as part of the Microsoft Data Access Components (MDAC) SDK can be used with the OLE DB Provider for AS/400 and VSAM. To use the OLE DB Provider for AS/400 and VSAM, you must specify **SNAOLEDB** for the provider name.

For more information about how to use the Data Description utility, see the Host Integration Server SDK documentation.

See Also

Concepts

Host File Transfer

Other Resources

Data Integration User's Guide

How to Add or Configure a Data Link for Windows

The following procedure shows the process for adding or configuring a data link.

To add or configure a data link for Windows

1. Click **Start**, point to **Programs**, and then point to **Host Integration Server**.
2. Point to **Data Integration**, and then click **OLE DB Data Source**.

   The **Data Link Properties** dialog box appears.
3. Configure the data source information for the selected provider.

   Click **Help** for more information.
4. Click **OK** to save the data link.

See Also

Concepts

Host File Transfer

Data Description

Other Resources

Data Access

Network Integration User's Guide

The topics in this section provide an overview of network communication in a Host Integration Server environment and describe the services you can use to manage it.

In This Section

IP-DLC Link Service

SNA Service

Host Print Service

TN Service

Active Directory Services

Host Configuration

Applications and Tools

IP-DLC Link Service

This section describes the configuration and use of the IP-DLC Link Service.

In This Section

Introduction to the IP-DLC Link Service

This section describes the overall architecture of the Host Integration Server system with IP-DLC link service functionality.

IP-DLC is a Host Integration Server feature that provides SNA connectivity for applications using dependent and independent sessions over a native IP network. It implements the HPR/IP protocol, which is also known as HPR over IP or Enterprise Extender. Each SNA packet is transmitted natively across the IP network as a UDP datagram.

In This Section

System Overview

This topic outlines key points of the Host Integration Server implementation of the IP-DLC link service.

- The IP-DLC link service is configured and managed by the Host Integration Server administrator as for any other link service.

- To provide the support required for transporting dependent LU sessions across an APPN network, the IP-DLC link service runs the DLUR feature as defined by the SNA/APPN DLUR Architecture Reference. This provides dependent LU session support to a CS/390 host running the matching DLUS feature.

- The IP-DLC link service operates as a Branch Network Node (BrNN) as defined in the SNA/APPN Branch Extender Architecture Reference. When large numbers of Host Integration Server systems are connected to a mainframe, the Branch Network Node configuration ensures that the overhead of topology and network search traffic on the Host Integration Server and mainframe links is minimal—comparable to using Host Integration Server without the IP-DLC link service.

- Multiple IP-DLC link services can be configured, one for each local IP address on the Host Integration Server system.

- Multiple IP-DLC connections are supported for each IP-DLC link service. There is one IP-DLC connection for each PU local to Host Integration Server.

- Remote independent LUs are associated with an IP-DLC link service instead of a connection. All remote independent LUs accessible through an IP-DLC link service are associated with the single peer connection for the IP-DLC link service. All independent LU traffic is routed by the APPN network and may traverse any active HPR/IP link.

See Also
Concepts
Supported Features
Scalability
Key Limitations
IP-DLC Link Service Concepts and Terminology

Supported Features

This topic lists the features that are supported over the IP-DLC link service.

Base SNA Features

The following existing Host Integration Server SNA features are supported over the IP-DLC link service:

- LU types 0, 1, 2, 3 and 6.2 (dependent and independent)
- LUA, FMI, APPC, and CPI-C APIs
- Dynamically Defined Dependent LUs (DDDLUs)
- Dynamic addition of local LUs, remote LUs, and connections
- Incoming and outgoing calls
- Connection Activation at Server startup, on demand or by administrator
- SNA data compression
- PU concentration with the upstream link over IP-DLC
- NetView RUNCMD/Alerts (note that the IP-DLC link service does not generate alerts)

Because the preceding SNA features are supported, it follows that the following applications are also supported over the IP-DLC link service:

- Host Integration Server-compatible 3270 emulators
- Host Integration Server-compatible 5250 emulators
- APPC - 3270 Session Viewer (including the LU-LU Test feature)
- Host Print Service
- Data Integration Services
- Local and remote administration
- TN3270 server
- TN5250 server
- MSMQ-MQSeries Bridge
- Transaction Integrator

Fault Tolerance Features

Note that because the IP-DLC link service uses the APPN and HPR/IP protocols, it is automatically able to take advantage of the following fault tolerance features:

- Ability to reroute sessions around a failure in the network provided that an alternative route exists.

- Ability for mainframe applications to be moved to a different processor with little or no impact to users when system or application failures occur on the mainframe.

Load Balancing Features

Load balancing for a single IP-DLC link service, with a single local IP address, over multiple adapter cards can be achieved using NIC Teaming at the MAC layer. The IP-DLC link service handles frames arriving out of sequence.

Security Features

Because the IP-DLC link service uses UDP/IP, the Windows IPSec feature can be used to provide end-to-end data security over the IP-DLC link service. Windows handles configuration of IPSec independently. No specific configuration is required for the IP-DLC link service.

Diagnostic Features

The diagnostics provided by the IP-DLC link service are used through the Host Integration Server Tracer Initiator and Trace Viewer facilities.

See Also

Concepts

System Overview
Scalability
Key Limitations
IP-DLC Link Service Concepts and Terminology

Scalability

The IP-DLC link service supports the Host Integration Server capacity of 30,000 simultaneous host sessions per server.

Note
Four nodes are required to achieve 30,000 simultaneous host sessions and a single node supports a maximum of 15,000 simultaneous host sessions.

See Also

Concepts

System Overview
Supported Features
Key Limitations
IP-DLC Link Service Concepts and Terminology

Key Limitations

The key limitations with the IP-DLC link service implementation are as follows:

- The IP-DLC link service cannot be run as a distributed link service (DLS).
- The PU Passthrough and Downstream connections are not supported over IP-DLC connections. It is not possible to have a one-to-one correspondence between upstream and downstream messages where the upstream connection is an IP-DLC connection.
- Each IP-DLC link service must use a different CP name from the SNA node service.
- Each IP-DLC link service requires a unique local IP address. If multiple IP-DLC link services are required, each must have its own unique local IP address.
- A single IP-DLC link service cannot be shared by multiple SNA node services. Each SNA node service must use a different IP-DLC link service for IP-DLC connectivity.

See Also
Reference
Key Limitations
Concepts
System Overview
Supported Features
Scalability

IP-DLC Link Service Concepts and Terminology

The following is a brief introduction to the terminology and concepts that are referred to in this section, including APPN and HPR.

Session

A session is a logical connection between two network accessible units (NAUs). The most common example of an NAU is a logical unit (LU) (for more information, see Logical Unit (LU) later in this topic).

Physical Unit (PU)

The component that manages and monitors the resources (such as attached links and adjacent link stations) associated with a node. This term applies to non-APPN nodes only.

Logical Unit (LU)

A logical unit (LU) is a port through which an application or end user accesses the SNA network to communicate with another application or end user. An LU may be capable of supporting many sessions with other LUs.

There are two types of LUs:

- Dependent LUs require assistance from a mainframe to establish a session with another LU. These are also sometimes referred to as old LUs.
- Independent LUs can establish a session with another LU without the assistance of a mainframe.

APPN

Advanced Peer-to-Peer Networking (APPN) is a network architecture that supports distributed network control. It makes networks easier to configure and use, provides centralized network management, and supports flexible connectivity.

APPN Nodes

APPN nodes include systems of various sizes, such as mainframes using CS/390, Solaris servers running DCLs SNAP-IX, PCs running IBM Communications Server for Windows NT/2000 and IBM AS/400.

In an APPN network, nodes can be one of the following types:

- Network Nodes (NN)
- End Nodes (EN)
- Branch Network Nodes (BrNN)
- Low-entry networking nodes (LEN nodes)

Each node in an APPN network is connected to at least one other node in the APPN network. Where supported, CP-CP (Control Point to Control Point) sessions are established over these connections to adjacent nodes (nodes in the same network that can establish direct connections without going through a third node). CP-CP sessions are used to exchange network topology information, request the location of network resources, and manage sessions. All of the nodes in an APPN network share a common network name.

Network Node

A Network Node provides distributed directory and routing services for all LUs in its domain, where its domain is all directly attached End Nodes and LEN nodes that are using the services of the Network Node. The Network Node is referred to as the Network Node Server (NNS) for those directly attached End Nodes and LEN nodes.

A Network Node provides the following services:

- LU-LU session services for its local LUs.
- Directory searches and route selection for all LUs in its domain.

- Intermediate session routing for sessions between LUs on different nodes.

- Routing for Management Services (MS) data, such as alerts, between a served End Node or LEN node and an MS focal point.

End Node

An End Node is an end point in an APPN network. It maintains directory information only for local resources. An APPN End Node can independently establish sessions between local LUs and LUs on adjacent nodes. For sessions with LUs on nodes not directly connected to the End Node, an End Node requests routing and directory information from its Network Node Server using CP-CP sessions.

End Nodes can register their local LUs with their Network Node Server. This capability means the network operator at the Network Node Server does not have to predefine the names of all LUs on the attached End Nodes to which the Network Node provides services.

An End Node can be attached to multiple network nodes, but it can have CP-CP sessions active with only one Network Node at a time: its Network Node Server. The other Network Nodes can be used only to provide intermediate routing for the end node or as substitute Network Node servers if the main Network Node Server becomes unavailable.

An End Node can also have a direct connection to another End Node or LEN node, but CP-CP sessions are never established between the two nodes.

LEN Node

A LEN Node is a type 2.1 node that uses independent LU 6.2 protocols, but does not support CP-CP sessions. It can be connected to a Network Node or End Node but does not support APPN functions. The existing SNA node of Host Integration Server is a LEN node.

A Network Node can provide routing services for an attached LEN node, enabling the LEN node to participate in an APPN network without requiring links to be defined between the LEN node and all of the nodes in the APPN network.

LUs in the APPN network with which the LEN node may want to establish sessions must be defined to the LEN node as if they reside on the LEN node's Network Node server. The LEN node establishes sessions with LUs defined to be contacted through its Network Node Server. The Network Node routes the session through the APPN network to the node in the network where the LU actually resides.

LUs on the LEN node must be predefined to the Network Node that serves the LEN node. LU resources on LEN nodes (unlike those on End Nodes) cannot be registered on the Network Node Server by the LEN node.

When a LEN node's only link is to an End Node, the LEN node can communicate only with LUs on the End Node through the direct link between the two nodes. This is because an End Node cannot provide intermediate routing.

Branch Network Node

The Branch Network Node (BrNN) combines the functions of a Network Node and an End Node. As the name implies, a BrNN can be used to subdivide a network into a backbone network and attached branch networks. The BrNN provides the following functions:

- To the backbone network, the BrNN appears as an End Node, connected to its Network Node Server (NNS) in the backbone network.

- The nodes in the backbone network are not aware of the nodes within the branch, reducing the amount of topology information that must be stored.

- Because the BrNN appears as an End Node, it does not receive topology information from the backbone network (topology information is transmitted only between Network Nodes) reducing the amount of network overhead traffic flowing into the branch network.The BrNN registers all resources in the branch with its NNS as though they were located on the BrNN itself. This means that the nodes in the backbone network can locate resources in the branch without having to be aware of the separate nodes in the branch.

- To the branch network, the BrNN appears as a Network Node, acting as the NNS for End Nodes and LEN Nodes in the branch.

High Performance Routing

High Performance Routing (HPR) is an extension of the APPN architecture. HPR provides the following functions:

- Rapid Transport Protocol (RTP) minimizes processing cycles and storage requirements for routing network layer packets through intermediate nodes on a session route.
- Automatic network routing (ANR) enables APPN networks to automatically reroute sessions if a portion of the originally computed route fails.

Dependent LU Requester/Server

Dependent LU Requester (DLUR) function enables sessions for dependent LUs to reside on remote nodes across an APPN network, instead of requiring a direct connection to the host.

DLUR works in conjunction with Dependent LU Server (DLUS) at the host. Together, they route sessions across the network from dependent LUs in the APPN network to the DLUS host. The route to the host can span multiple nodes and can take advantage of APPN's network management, dynamic resource location, and route calculation facilities.

If the local node is a Network Node, dependent LUs on downstream computers can also use pass-through DLUR, in the same way that LUs internal to the node do, to access the host across the network.

See Also

Concepts

System Overview
Supported Features
Scalability
Key Limitations

Managing IP-DLC Link Services

This section gives procedures for creating, configuring, viewing, and deleting IP-DLC link services.

In This Section

Creating an IP-DLC Link Service

You create and configure an IP-DLC link service as you would create any other link service.

To create an IP-DLC link service

1. In the SNA Manager, locate the computer on which you want to create the IP-DLC link service.
2. Right-click the **Link Services** folder under that computer.
3. On the context menu, click **New/Link Service.** The **Insert Link Service** dialog box appears.
4. From the list of available link services, select **IP-DLC Link Service**.
5. Click **Add** to load the **IP-DLC Link Service** properties dialog box.
6. To configure the new link service, follow the steps for Configuring an IP-DLC Link Service.

See Also

Other Resources

Managing IP-DLC Link Services

Viewing Link Services

As with other link services, you can view IP-DLC link services in the list view of the **Link Services** folder.

To view link services

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer with the link services that you want to view.
2. Expand the list for that computer.
3. Select the **Link Services** folder. The link services for that computer will appear in the right pane.

See Also

Other Resources

Managing IP-DLC Link Services

Viewing Link Service Properties

As with other link services, you can view IP-DLC link service properties by using the context menu in the list view of the **Link Services** folder.

To view link service properties

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer with the link services that you want to view.
2. Expand the list for that computer.
3. Select the **Link Services** folder.

   The link services for that computer appear in the right pane.
4. Right-click the appropriate link service, and then click **Properties**. The **Link Service Properties** page appears.
5. Click **Configure**.

   The properties for this IP-DLC link service appear, and can be viewed or edited.

See Also

Other Resources

Managing IP-DLC Link Services

Deleting Link Services

As with other link services, you can delete an IP-DLC link service in the MMC snap-in.

Note
When you delete a link service, all related connections will no longer be associated with any link service. A user might associate such a connection with any other IP-DLC link service that is associated with the node. Deleting a node associated with a link service will break association between these items.

To delete a link service

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer with the link services that you want to delete.
2. Expand the list for that computer.
3. Right-click the appropriate link service, and then click **Delete**.

See Also

Other Resources

Managing IP-DLC Link Services

Link Service and the LnkCfg Utility

LnkCfg is a useful command-line utility for deploying and managing link services. The format of the command line for configuring the link service is specified as follows.

LINKCFG LINKSVC "Title"

/SERVER:servername

/LSTYPE:"IP-DLC Link Service"

/PRIMARYNNS:NNSServer

/BACKUPNNS:NNSServer

[/LOCALADDRESS:ipaddress] or [/ADAPTER:adaptername]

/NETWORKNAME:networkname

/CPNAME:name

/NODEID:"xxx.xxxxx"

/LENNODE:lennode

The following table describes the command-line parameters.

Property	Description	Content
"Title"	Title of the link service.	1-128 characters.
/SERVER:servername	Name of the server.	Valid server name.
/PRIMARYNNS:NNSServer	Primary network node server.	DNS name or IP address.
/BACKUPNNS:NNSServer	Backup network node server.	DNS name or IP address.
/ADAPTER:adaptername	Name of the local adapter. /ADAPTER and /LOCALADDRESS arguments should not be used together.	Name of the physical or logical adapter on the computer.
/LOCALADDRESS:ipaddress	Local address. /ADAPTER and /LOCALADDRESS arguments should not be used together.	Valid IP address or server name.
/NETWORKNAME:networkname	Network name of the Branch Network Node implemented by the link service.	1-8 characters SNA Type A string.
/CPNAME:name	Control point name of the Branch Network Node implemented by the link service.	1-8 characters SNA Type A string.
/NODEID:"xxx.xxxxx"	Identity of the Branch Network Node implemented by the link service.	String in format HHH.HHHHH where H is a hexadecimal digit.
/LENNODE:lennode	Name of the associated LEN node.	Name of a LEN node deployed on the local computer.

See Also

Other Resources

Managing IP-DLC Link Services

Managing IP-DLC Link Service Connections

This section gives procedures for creating, configuring, viewing, and deleting IP-DLC link service connections.

In This Section

Creating an IP-DLC Connection

You create and configure an IP-DLC link service connection as you would create any other link service connection.

To create an IP-DLC connection

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer on which you want to create the IP-DLC link service connection.
2. Under that computer, expand **SNA Service**.
3. Right-click the **Connections** folder under that computer.
4. On the context menu, click **New**, and then click **IP-DLC**.
5. To configure the new link service connection, follow the steps for Configuring an IP-DLC Connection.

See Also

Tasks

Viewing Connections

Viewing Link Service Properties

Concepts

Defining Dependent LUs

Connections and the SnaCfg Utility

Secure Deployment of the IP-DLC Link Service

Viewing Connections

As with other connections, you can view IP-DLC link service connections in the list view of the **Connections** folder.

To view connections

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer with the link services that you want to view.
2. Expand the list for that computer, expand **SNA Service**, and expand **Connections**. The link service connections for that computer will appear in the right pane.

See Also

Tasks

Creating an IP-DLC Connection

Viewing Link Service Properties

Concepts

Defining Dependent LUs

Connections and the SnaCfg Utility

Secure Deployment of the IP-DLC Link Service

Viewing Connection Properties

As with other link services, you can view IP-DLC link service properties by using the context menu in the list view of the **Link Services** folder.

To view connection properties

1. In the scope pane of the Host Integration Server MMC snap-in, locate the computer with the link service connections that you want to view.
2. Expand the list for that computer, expand **SNA Service**, and expand **Connections**. The link service connections for that computer will appear in the right pane.
3. Right-click the appropriate link service connection, and then click **Properties**. The **Connection Property** page will appear, and the properties for this connection can be viewed or edited.

See Also

Tasks

Creating an IP-DLC Connection
Viewing Connections

Concepts

Defining Dependent LUs
Connections and the SnaCfg Utility
Secure Deployment of the IP-DLC Link Service

Defining Dependent LUs

You can configure dependent LUs (both 3270 and LUA) as you would with any other link service.

See Also

Tasks

Creating an IP-DLC Connection

Viewing Connections

Concepts

Connections and the SnaCfg Utility

Secure Deployment of the IP-DLC Link Service

Connections and the SnaCfg Utility

SnaCfg is a useful command-line utility for deploying and managing SNA Server configurations.

The following table describes the command-line parameters.

Property	Description	Validation
/conntype:type	Connection type.	IP-DLC.
/RemoteAddress:adr	Address of the remote DLUS service. This property is not accessible through the user interface. It provides a way of establishing a direct connection with DLUS rather than routing the connection through the NNS.	Valid IP address or DNS name.
/PrimNetworkName:name	Network name of the primary DLUS server.	1–8 characters, SNA Type A string.
/PrimCPName:name	Control point name of the primary DLUS server.	1–8 characters, SNA Type A string.
/BackupNetworkName:name	Network name of the backup DLUS server.	1–8 characters, SNA Type A string.
/BackupCPName:name	Control point name of the backup DLUS server.	1–8 characters, SNA Type A string.
/DLURRetryType:N	DLUR retry type.	0 indicates none 1 indicates infinite 2 indicates limited
/DLURRetryLimit:N	DLUR retry limit. This parameter is invalid unless the DLUR retry type is set to limited.	1–65534
/DLURRetryDelay:N	Delay after a DLUR retry. This parameter is invalid unless the DLUR retry type is set to limited.	1–65535
/RetryLimit:N	Number of the connection retries.	0 indicates unlimited 1–65534 number of retries
/RetryDelay:N	Delay after a connection retry.	0–327670, must be a factor of 5.
/XIDFormat:N	XID type.	Set to: 1 – "Format 3"

/RemoteNetName:name	This value is always set to the network name of the IP-DLC link service.	Must be left blank for a new connection.
/RemoteCPName:name	This value is always set to the control point name of the IP-DLC link service.	Must be left blank for a new connection.

See Also

Tasks

Creating an IP-DLC Connection

Viewing Connections

Concepts

Defining Dependent LUs

Secure Deployment of the IP-DLC Link Service

Secure Deployment of the IP-DLC Link Service

The IP-DLC link service takes advantage of the entire Host Integration Server secure deployment feature set, especially the following:

- A typical IP-DLC link service deployment scenario may use Internet Protocol security (IPSec), a firewall, or a virtual private network (VPN) as a security barrier between the Host Integration Server computer and the remote node. (A VPN is advisable if the HPR/IP link spans an insecure IP network.)

- Incoming frames from IP addresses that are not already defined in Host Integration Server are rejected. This isolates the main SNA node service from such attack.

- The IP-DLC link service checks the length of incoming datagrams before copying the data into internal data areas, and discards any that are too long.

- All configuration data—whether from registry entries, COM.cfg, or the SNA node service—is validated for correct value when first accessed. Particular consideration is given to validating lengths to avoid buffer overruns. If any errors are found, the IP-DLC link service logs an event and terminates.

See Also
Tasks
Creating an IP-DLC Connection
Viewing Connections
Concepts
Defining Dependent LUs
Connections and the SnaCfg Utility

SNA Service

The core of the power of Host Integration Server is its ability to provide a wide range of host connectivity services. Host Integration Server uses client/server architecture to distribute communications processing. This architecture maximizes the power of the host, Host Integration Server, and individual client computers. Within the context of Host Integration Server, a connection is the data communication path between a Host Integration Server computer and an IBM host (mainframe or AS/400). A connection is what makes it possible for a client computer on a local area network (LAN), using standard LAN protocols, to communicate with a host by means of one or more Host Integration Server computers. Each server supports up to 3,000 users operating 30,000 concurrent sessions, 8,000 split stock users per node, four nodes per server.

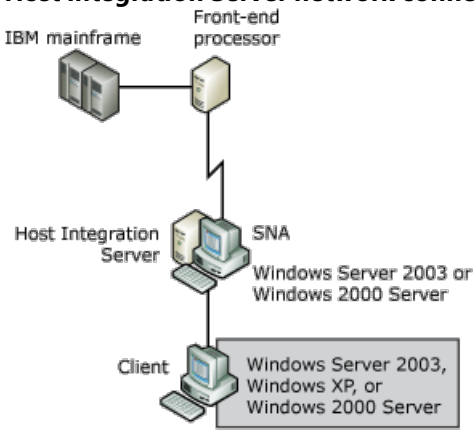
In This Section

Communication Between Host Integration Server Computers and a Host Computer

Within the context of Host Integration Server, a connection is the data communication path between a Host Integration Server computer and an IBM host (mainframe or AS/400). A connection corresponds to a physical unit (PU) definition on a mainframe or an APPC controller definition on an AS/400. The connection is what makes it possible for a personal computer on the LAN to communicate with a host by means of a Host Integration Server computer. Note that SNA connections do not use BISYNC, an older IBM standard for communications.

The following figure shows a Host Integration Server network connected to an IBM mainframe.

Host Integration Server network connected to IBM mainframe



For each physical adapter or connection, an appropriate link service is installed and configured within Host Integration Server. The link service is a Windows server service or device driver that is used to control server-to-host communication adapters supported by Host Integration Server. The link service provides the SNA data link-level protocol used by the Host Integration Server computer to communicate with the host.

After being configured, the link service is available for use not only on the configured Host Integration Server, but also on any Host Integration Server computer in the subdomain using the Distributed Link Service feature of Host Integration Server.

After a link service is configured, you can create connections. A link service may support multiple links to one or more hosts.

In SNA terms, the combination of a connection and the link service it uses is equivalent to a PU. In hierarchical SNA networks, Host Integration Server provides PU 2.0 functionality. For peer-oriented SNA networks, Host Integration Server provides PU 2.1 LEN node functionality.

See Also

Other Resources

SNA Service

Physical Unit (PU)

Although the name physical unit (PU) strongly implies a hardware component, a PU is an IBM naming convention for hardware and software combinations that perform a specific task in an SNA network. For example, PU 5 represents not only the mainframe itself, but also the mainframe software components (such as SSCP and VTAM) within the SNA network.

The following table lists the SNA defined PU types and a brief description of each.

PU number	Description
PU 1	Terminal Controller (IBM 6670, 3767)
PU 2	Cluster controller running configuration support programs (IBM 3174, 3274, 4701, 4702)
PU 2.1	Peer-to-Peer (APPN), used primarily in AS/400 networks
PU 4	IBM Front End Processor, usually running ACF and the Network Control Program (IBM 3754, 3725, 3720, 3745, 3746)
PU 5	IBM Host (mainframe) system

See Also

Other Resources

SNA Service

Logical Unit (LU)

A logical unit (LU) is a configurable unit of software that contains the information needed to specify the type of communications session with the host computer or peer system. Thus, an LU is a point of access to the Host Integration Server network. There are several types of LUs:

- APPC
- 3270
- Downstream
- Logical unit application (LUA)

LUs represent a set of functions that manage the exchange of data between users and applications, acting as intermediaries between the user and the network. Host Integration Server protocols identify several LUs on host computers that represent specific functions. The following table is a list of LU numbers and a brief functional description.

LU number	Description
LU 0 (also LU/A or LUA, logical unit application)	General purpose LU for development of specialized applications such as TN3270. Used for program-to-program communications in hierarchical networks.
LU 1	LU 1 handles the transmission of printer data to network and system printers in SNA character string format. IBM 3287-type printers and Host Integration Server Host Print service that allows you to redirect print data streams to LAN printers. Used to communicate with multiple-device terminals.
LU 2	IBM monochrome terminals 3278 (3270) IBM Color graphic terminals 3279/3179 Host Integration Server displays
LU 3	IBM 3284-style printers and Host Integration Server Host Print service, which allows you to redirect print data streams to LAN printers. LU 3 is a simple printing protocol that uses the 3270 data stream format to communicate with a single printer.
LU 4	IBM 6670 information distributor and is not supported through Host Integration Server.
LU 5	Not defined.
LU 6	LU 6.2 is most common revision. IBM 5250 devices and Host Integration Server local and remote APPC LUs. Provides peer-to-peer communication through Advanced Program-to-Program Communication (APPC) and (Common Programming Interface for Communications (CPI-C).
LU 7	IBM 5250 display terminal. Used mainly on System 3.x but not on AS/400 systems.

On Host Integration Server computers, you can configure LUs to emulate the 3270 data streams needed to communicate with the mainframes or 5250 data streams for AS/400 systems.

A 3270 LU is a dependent LU that requires the mainframe to function. It has a fixed designation, such as a display LU, printer LU, logical unit application (LUA) LU, or downstream LU. Each type of LU has a specific number assigned to it. A 3270 LU on Host Integration Server has an LU number, and a corresponding resource must be defined on the host computer. The 3270 LU name is arbitrary and does not have to match the name of the resource on the host.

Additional information about APPC is available later in this section.

See Also

Other Resources

SNA Service

Choosing a Connection Type

A Host Integration Server computer can use any of the types of connections described in this section. To choose a connection type for your servers, you should first contact the host administrator and find out the type of connection available to the mainframe or AS/400. If more than one type is available, choose a type based on comparisons of cost and speed.

For demonstration purposes, you can install the Demo SDLC link service, which receives messages from Host Integration Server and responds to them itself from a prerecorded script.

The various adapters in a computer must be configured to work together so that there are no interrupt, port address, or direct memory access (DMA) conflicts. When you install a new adapter in your computer, you may need to study the configuration of the new and old adapters to make sure that there are no conflicts.

The following connection types are available:

- 802.2 Data Link Control (DLC 802.2)

  Token ring, Ethernet, or Fiber Distributed Data Interface (FDDI) connections use the IEEE 802.2 protocol. With a mainframe, an 802.2 connection goes to a 37*xx* front-end processor (FEP) or a 3174 communications controller (or, rarely, to an adapter in the mainframe). With an AS/400 system, an 802.2 connection goes directly to the AS/400.

  These connections are generally faster than other connections, except for channel connections. The following table provides details.

Type of 802.2 connection	**Common speeds**
Token ring	4 megabits per second (Mbps) or 16 Mbps.
Ethernet/Fast Ethernet	10–100 Mbps.
FDDI	100 Mbps (or more); however, the FDDI line communicates through a front-end processor that is channel-attached to the mainframe, and this may limit the overall speed of communications.

  DLC 802.2 also has certain limitations. The 802.2 link service cannot support more than one 802.2 connection to the same MAC address and destination SAP (that is, to the same host) at the same time, whether the connections originate from the local or a remote SNA Server computer. The 802.2 protocol specification does not permit two distinct connections to have the same Source MAC, Source SAP, Destination MAC, and Destination SAP. The 802.2 link service is bound to a particular LAN adapter and source SAP value, and therefore uses the same source MAC and SAP for all connections. To support multiple connections, you can:

  - Connect to different network adapters on the host.
  - Configure multiple 802.2 link services, all bound to the same LAN adapter but specifying different source SAPs.
  - Support multiple connections to a given host through the same 802.2 link service. Verify that the host is configured to accept 802.2 connections on multiple SAPs. This can be configured to mainframe FEPs (but is rarely used). AS/400s support multiple SAPs by default. These SAP values are configured as the remote SAP value on the Host Integration Server connection, which is configured on the Host Integration Server computer using the distributed link service.

  This limitation applies to connections based on the 802.2 link service in general.

  When you make an 802.2 link service available to distributed link service on other Host Integration Server computers, it cannot be used by the local Host Integration Server computer, and is not visible in SNA Manager Console.

- Synchronous Data Link Control (SDLC)

SDLC uses a standard phone line, which can be leased or switched and can be point-to-point or multi-drop. An SDLC line is used with a modem or other type of data circuit -terminating equipment (DCE) at each end. With a mainframe, an SDLC line travels through a modem or other DCE to a 37*xx* front-end processor (FEP), 3174 communications controller, or integrated synchronous adapter. With an AS/400 system, an SDLC line goes through a modem or other DCE, and then directly to the AS/400.

An SDLC connection is slower than an 802.2 or channel connection. Common speeds for SDLC connections are listed in the following table.

Type of transmission	**Common speeds**
Analog (conventional phone line)	9600–56000 bits per second
Digital Data System (DDS)	56 kilobits per second (kbps) (sometimes 64 kbps)
Integrated Services Digital Network (ISDN)	56 kbps (can be more)
T1 carrier system (digital)	1.544 megabits per second (Mbps)
E1 carrier system (digital)	2.048 Mbps

SDLC connections are useful for wide-area connections between geographically disparate locations, or when bandwidth and usage requirements are low. Because of these factors, SDLC is ideally suited for branch-type deployment strategies.

Host Integration Server supports SDLC connections using the link support that is included with it or through an SDLC link service available through various third-party vendors. Not all supported SDLC adapters support all link speeds listed in preceding table.

- Distributed Link Service

  The distributed link service feature provides a method for a Host Integration Server computer to connect to a host using a link service installed on a different Host Integration Server computer. The network connecting SNA Server computers need not support any SNA link level protocol such as DLC 802.2 or SDLC.

  The distributed link service is configured in two parts: by installing the distributed link service on one SNA Server computer and by marking a real link service on another computer as distributable. The distributed link service acts as a proxy for sharing the distributable link service. It supports load balancing across multiple, distributed link services. It also supports hot backup because the distributed link service can select alternate remote servers when a remote link fails. It allows a branch SNA Server computer to connect to the host over a wide area network (WAN). This supports only routable internetworking protocols such as TCP/IP, rather than requiring an SNA WAN protocol such as SDLC or bridged DLC.

  Distributed link services provide the following benefits:

  - The branch-based Host Integration Server computers provide split-stack SNA gateway service for local client computers, simplifying configuration of the client computers. This conserves PUs and concentrates traffic on behalf of several client computers, which saves valuable network bandwidth.

  - The branch-based Host Integration Server computer communicates with the central-site Host Integration Server computers through a native TCP/IP connection, which eliminates DLC 802.2 time-out problems associated with traditional SNA encapsulation methods.

  - Because the central site servers provide the equivalent of PU pass-through service for the branch-based servers, the host operator sees each branch-based server as a PU and can manage the branches through standard NetView alerts and RunCmds.

  - The branch-based servers can connect to the host through multiple centralized servers, load-balancing among the multiple central-site servers at connect time.

- Should a central-site server fail for any reason, the branch-based servers will automatically establish a new connection through an alternate centralized server for hot backup.

- Should the WAN fail for any reason, the branch-based Host Integration Server computers can be configured to connect to the host through a direct dial-up SDLC connection as a backup that will be activated automatically upon the host connection failure. It is actually activated upon the first request after a failure.

- The routers at the branches only need to route TCP/IP, which provides for simplicity of WAN management and cost savings.

- Because the branch-based servers rely on WAN services provided by the leading networking vendors, the distributed link service will work over all existing and future WAN technologies, including leased lines, X.25, frame relay, and ATM networks.

- Unlike other native TCP/IP solutions such as TN3270, the distributed link service is not limited in SNA functionality. Each branch-based Host Integration Server computer provides full SNA access for the local personal computers, including PU 2.0, PU 2.1, and APPN LEN service, as well as LU 0, LU 1, LU 2, LU 3, and LU 6.2 support. Both mainframe and AS/400 access are supported.

- Although the majority of customers are moving to TCP/IP networks, the distributed link service also fully supports IPX protocols.

When Host Integration Server is set up to use more than one client/server protocol, distributed link service first attempts to establish communication between the servers using TCP/IP, if available. If that fails, distributed link service next attempts to connect using IPX/SPX. To designate a protocol other than TCP/IP as the primary protocol to be used by distributed link service, use the Registry Editor to add or modify the values in the Windows registry. For an example of the registry entry, see the *Host Integration Server Administrators Reference*.

By default, when you make a link service available to distributed link service on other servers, it will permit a connection from distributed link service running on any remote server that is configured to connect to it. This is similar to the level of access control permitted by traditional pass-through gateways. Any downstream device that is configured for the correct MAC address can use the gateway.

However, you can restrict access to the distributed link service, requiring the remote distributed link service to provide a valid Windows server username and password. The procedure requires coordination between the upstream and remote servers.

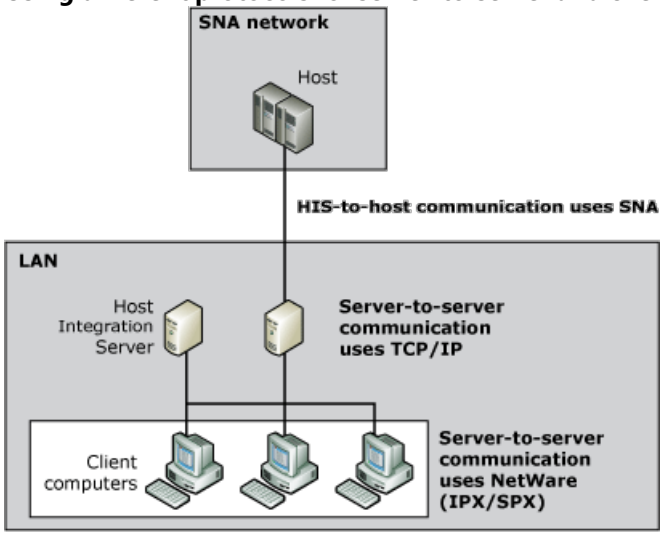
See Also
Other Resources
SNA Service

Communication Between Multiple Host Integration Server Computers

Host Integration Server computers communicate with each other using mail slot or datagram broadcast messages. They use the SnaBase service to keep track of names of servers, client computers, and available transaction programs (TPs), which are the programs used for communication through Advanced Program-to-Program Communications (APPC) or Common Programming Interface for Communications (CPI-C).

Microsoft Windows Server 2003 and Windows 2000 Server support multiple protocols simultaneously, but there is no need to send the same Host Integration Server broadcasts over all available protocols. Using the SNA Manager Console, you can specify the network protocol for server broadcasts. You must make sure that one protocol is available on all Host Integration Server computers in the SNA subdomain, and use that protocol for server broadcasts. For more information, see Configuring a Server Broadcast. The following figure shows a network in which TCP/IP is used for server-to-server communications and IPX/SPX is used for client-to-server communications.

Using different protocols for server-to-server and client-to-server communications



Separate protocol used for server-to-server and client-to-server communications

It is recommended that you use only one protocol for server broadcasts. Using multiple server-to-server transport protocols can add to network overhead because every server broadcast must be sent out through all the protocols selected.

Server broadcasts need to be configured only once for a subdomain; the configuration affects all servers in the subdomain.

The mean time between server broadcasts is specified in number of seconds, with the default being 60 seconds. Specifying a smaller value brings heavier demands on the network because the broadcasts occur more often.

See Also

Other Resources

SNA Service

Configuring a Server Broadcast

You can configure a server broadcast to ensure that your servers use the appropriate protocols for your network configuration.

To configure a server broadcast

1. In the SNA Manager tree, select the subdomain that you want to configure.
2. On the **Action** menu, click **Properties**.
3. Click the **Server Broadcasts** tab, fill in the options, and then click **OK** to exit.
4. On the **Action** menu, click **Save configuration**.

See Also

Other Resources

SNA Service

Communication Between Host Integration Server Computers and Client Computers

Client computers locate the resources of the Host Integration Server computer in either one of two ways:

- Sponsor connection
- Active Directory directory service

A sponsor connection is created when a client computer makes a request to communicate with a Host Integration Server computer and an initial connection is established. The first Host Integration Server computer to respond to the request (called the sponsor computer) provides the client computer with all the names of Host Integration Server computers in the subdomain. From these names, the client computer locates an available LU or LU pool.

In a similar way, a client computer that is configured to use Active Directory makes a request to the Active Directory database. Active Directory responds to the request and provides the client computer with all the names of Host Integration Server computers in the Organizational Unit.

For client/server protocols to operate properly with Host Integration Server, both client computers and servers must have the network software and the Host Integration Server software installed properly. The network protocols available for use by various client operating systems are listed in the following table.

Operating system on client computer	**Client/server protocols**
Windows Server 2003 or Windows 2000	Microsoft Networking (Named Pipes) Novell NetWare (IPX/SPX) TCP/IP

See Also

Other Resources

SNA Service

Host Integration Server Client and SNA Communications

When a client computer makes an SNA request, the client computer must direct that request to a domain or to one or more Host Integration Server computers. The appropriate way for a client computer to direct requests depends on the protocol used and the relative location of client computers and servers. The following table lists the ways that client computers direct SNA requests, and the information that will be requested by Setup during client software installation.

Network protocol used on client	**How the client directs SNA requests**	**Information to find out before running client setup**
Microsoft Networking	Either to the local domain (if the Host Integration Server computers are in the same domain as the client computer), or to one or two specific Host Integration Server computers in a remote domain.	Whether the client computer is in the same domain as the Host Integration Server computers, and if not, one or two names of Host Integration Server computers.
Novell NetWare (IPX/SPX)	To a specific domain. The Host Integration Server computers must be located in this domain for the client computer to locate them.	The name of the SNA subdomain in which the Host Integration Server computers are located.
TCP/IP	Either to a domain name or to a specific server name, depending on what is specified in the SNA Client Mode or Host Integration Server Location dialog box.	Contact your network administrator for additional information regarding a specific IP address for Host Integration Server.

When a client computer first makes a request for communication with the SNA network, it establishes an initial connection (the sponsor connection) with a Host Integration Server computer. The client computer then requests a logical unit (LU), either by name or by requesting an LU from a particular pool of LUs. If the request is for a particular LU, the sponsor server responds by providing the name of the Host Integration Server computer that contains the LU. If the request is for an LU pool, the sponsor server responds by providing the client computer with all the names of Host Integration Server computers in its subdomain that support the pool. From these names, the client computer picks a server at random.

Host Integration Server computers and client computers keep track of the names of servers in the SNA subdomain through the SnaBase service (formerly known as the network access protocol, or NAP). For client computers, the sponsor connections, along with the SnaBase provide a view into the server subdomain.

See Also

Other Resources

SNA Service

Host Print Service

Host Print service provides server-based 3270 and 5250 print emulation, allowing mainframe and AS/400 applications to print to a LAN printer supported by Microsoft Windows Server 2003 and Windows 2000. Host Integration Server Host Print service enables centralized control of LU print resources. You can administer all Host Print service functions using SNA Manager, including margin control, fonts, and characters per line. Host Print service also supports print-to-file with auto-incrementing file names. You can configure the file naming scheme for each printer LU.

The following three methods of printing host information are available:

- **Screen printing.** This method allows any 3270 or 5250 emulator to print what is on the display using print screen features of the client operating system. The printer output can be directed to a printer attached to the client computer or the network.

- **Client-redirected printing.** This method delivers an SNA host printer data stream (such as 3287) to the appropriate emulation application running on a Host Integration Client 2000 computer. The client software converts the data stream into data that can be printed locally or to a network printer.

- **Server-based redirected printing.** This method uses a server process to convert SNA host printer data streams into data that can be directed to a local or network printer.

In This Section

Using Host Print Services

Host Print service is a Windows Server service, and is found as SnaPrint in the Services utility in Windows Control Panel. To access network printers, the SnaPrint service must run in a domain account with administrative privileges. You must create this account before Host Integration Server Setup.

See Also

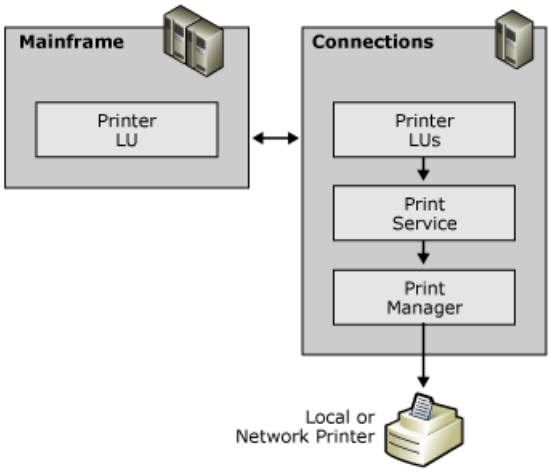
Other Resources

Host Print Service

Mainframe Printing

Mainframe printing supports both LU 1 and LU 3 data streams. It also includes pass-through support from the host for Intelligent Printer Data Stream (IPDS). Using IPDS, mainframe print jobs can be printed on LAN printers without changing either the print job or the host application.

Connections between mainframe printing and local printer



Host Print service provides print emulation for three LU types: LU 3, LU 1, and LU 6.2 (APPC). The LU type defines the characteristic of the host data stream. LU 3 and LU 1 printing use a 3270 data stream over a session to a mainframe. APPC printing uses a LU 6.2 data stream over a session to an IBM AS/400. The following sections provide an overview of LU 3 and LU 1 printing. APPC printing is discussed under the AS/400 section later in this section.

The Host Print service must run in a user account that has permission to access the defined network printer (or printers). For example, if you are printing to a printer attached to a Novell NetWare system, the Host Print service must run in an account that is recognized by NetWare.

See Also

Other Resources

Host Print Service

LU 3 Printing

LU 3 printing is the simplest of the three types of host printing. The LU 3 data stream closely resembles that for 3270-display emulation. It consists of a write command code and a write control character (WCC) followed by the print job. These print jobs contain printable characters and only four simple formatting orders (new line, form feed, carriage return, and end of medium). No other control of the print output is available. This form of printing is similar to printing a text file from a personal computer. If more print formatting is required, an LU 1 print session should be used. For more information, see the section LU 1 Printing.

In SNA Manager, the 3270 Print Session properties Page Layout tab includes specifications for the number of characters per line and the number of lines per inch. Host print jobs using the LU 3 data stream also provide defaults for these parameters. The defaults may be different depending on the settings.

For LU 3, the settings in the host print job normally override the Print Service session settings. However, if you leave the host settings at their defaults of 6 lines per inch and 132 columns per row, the Print service session settings are used.

After the defaults for lines per inch and page width have been determined, the Print service chooses a font for the print job that will accommodate this size print output on the paper loaded in the printer.

There are no transparent sections in LU 3 print jobs, so Transparency is ASCII does not apply. LU 3 print jobs do not have SCS codes, but do have NL, CR, FF, and LF characters and do have commands to allow you to position print data on the page. The Print service uses these to build an internal representation of the page before printing it out. When a PDT file is used, Print service uses the definitions of NL, CR, FF, and LF from the PDT file to allow it to format the printout correctly. If no PDT file is configured, Print service uses the Windows GDI.

Command Code

The write command codes are not unique values, but are identified by occupying the first byte in the Request Unit (the message format used in an SNA network, also known as an RU). In addition there can be only one command code per RU. For LU 3 printing, the most commonly used command code is Erase/Write '0xF5'. Read command codes, which would be normal for display sessions, are not valid for LU 3 printing, and will be rejected with a sense code of '1003'.

Write Control Character (WCC)

Following the write command code is the write control character (WCC). This byte is also identified by its position, second byte in the RU. With LU 3 printing, bits 2 and 3 of the WCC define the printout format.

Write Control Character (WCC) Codes

Bit	Explanation
0,1	Ignored by the printer.
2,3	Defines printout format.
none	00 NL or CR orders define the print line length, EM indicates the end of the message.
none	01 Indicates 40-character lines.
none	10 Indicates 64-character lines.
none	11 Indicates 80-character lines.
4	Start-printer bit.
6	Keyboard reset.
7	Reset MDT bit.

Format Control Orders

There are four control codes used only for printing known as Format Control Orders.

Format Control Orders

Abbreviation	Order	EBCDIC

NL	New Line	0x15
EM	End of Medium	0x19
FF	Form Feed	0x0C
CR	Carriage Return	0x0D

NL, CR, and EM are valid only when the write operation does not specify a line length format in the WCC byte. FF is valid in any write operation.

3270 Orders

The 3270 data stream can contain sequences, called 3270 Orders, which provide additional control functions. The two most commonly used in LU 3 printing are Set Buffer Address (SBA) and Repeat to Address (RA). Note that the buffer address used in these commands is relative to each write. The print buffer in LU 3 allows a maximum of 4 KB of data, and often only 2 KB. This may require multiple write commands to be sent for a full page of text. The first write command will start at the top of the page. Its first buffer address will also be at the top of the page. Subsequent writes will continue where the first left off. Their first buffer address will also start where the previous write ended, unless it was ended with a form feed. For the examples below, it is assumed that these are the first write commands.

SBA is indicated by a '0x11' followed by a two-byte buffer address. This order sets the cursor position to the location specified in the two-byte buffer address. In LU 3 printing, this sets the print position. The data following the SBA will be printed starting from this location. For example:

114040 Sets the print position to row 1 and column 1.

RA is indicated by a '0x3C' followed by a two-byte stop buffer position and the character to be repeated. This order causes a character to be repeated from the current buffer address up to but not including the stop buffer address specified in the RA. For example:

3C40D3C1 Repeats the character 'A' ('0xC1') to row 1 and column 20.

Data

For LU 3 printing the data or printable characters must have values between '0x40' and '0xFE'. The only valid values outside this range are the 3270 Orders.

Example: 15C1C2C3 Prints a new line followed by 'ABC'

Example: 1BC1C2C3 Rejected because '0x1B' is an invalid value

Sample Host Data

Following is sample data from a host along with an explanation of the data and resulting printout.

```
F5C81140 40151515 C1C2C3C4 15404040
E6E7E8E9 19
```

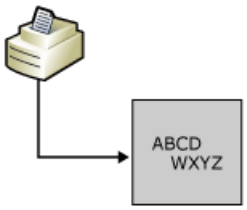
This sample data is analyzed in the following table.

3270 LU 3 Sample Data

Data	Interpretation
F5	Command code Write/Erase
C8	WCC with bits 2, 3 specifying that NL, EM and CR orders determine the print line length.
114040	SBA sets print position to row 1 column 1
15	New line

15	New line
15	New line
C1C2C3C4	EBCDIC hex values for ABCD
15	New line
404040	EBCDIC hex values for three spaces
E6E7E8E9	EBCDIC hex values for WXYZ
19	End Medium

Print output from the sample data in the preceding table. ABCD on top print line and WXYZ indented on lower line.



See Also

Other Resources

Host Print Service

LU 1 Printing

LU 1 printing allows the host to specify formatting for a print job. This is accomplished through the use of SNA Character String (SCS) control codes. The SCS codes encompass the 3270 format orders (FF, CR, and NL), as well as providing additional control codes to format the print output. Through the SCS codes, the host application can set the margins, characters per line, and lines per inch. In addition to the format of the print job, SCS code allows the host application to send a transparent section. By using an SCS code, the host can mark a section of data as transparent. This will cause the print emulator (the Host Print service in this case) to not scan this section for SCS control codes but to pass it to the print output untouched. Transparent sections are most commonly used to embed printer control codes, such as the HP PCL, in the print job. Unlike LU 3 printing, there is no write command code or WCC. The first byte of the RU is either an SCS code or data.

Data Stream Flags

The BIND sent by the host for a LU 1 session indicates what SCS control codes are valid for this session. These Data Stream Flags are set in byte 18 of the BIND.

Data Stream Flags

Bit	Value	Description
0		**Base SCS code support**
	0	**Base support** (NL & FF only)
	1	**Full Base** includes Base plus the following:
		BS (back space)
		CR (carriage return)
		LF (line feed)
		ENP (enable presentation)
		INP (inhibit presentation)
		HT (horizontal tab)
		VT (vertical tab)
1		**Set Horizontal Format** (SHF)
	0	not supported
	1	supported
2		**Set Vertical Format** (SVF)
	0	not supported
	1	supported
3		**Vertical Channel Select** (VCS)
	0	not supported
	1	supported

4		**Set Line Density** (SLD)
	0	not supported
	1	supported
5		**Reserved**
6		**Bell** (BEL)
	0	not supported
	1	supported
7		**Transparent** (TRN) & **Interchange Record Separator** (IRS)
	0	not supported
	1	supported

SCS Codes

The SCS control codes are fully documented in the IBM Host Print Guide (document number SC31-7145). All of the SCS control codes fall within the range of '0x00'–'0x3F.' These codes range from single byte codes, such as Subscript '0x38', to multiple byte codes followed by several parameters, such as Set Horizontal Format '0x2BC1...'

The following are some of the more common SCS control codes used.

Note
[L]=length and **(Abv)** represents one-byte parameters in the SCS control code.

Set Horizontal Format (SHF) — '0x2BC1[L](MPP)(LM)(RM)(T1)(. . .)(Tn)'

[L] — Length of the parameters, including the length byte

MPP — Maximum Presentation Position; defines the characters per line

LM — Left Margin; column value for the left most print position

RM — Right Margin; column value for the right most print position

T1 — Horizontal tab stop; column value for a tab stop

Tn — Additional tab stops, which can be added in any order

Example

2BC1068401840542

2BC1 — SHF

06 — length of the parameters is 6 bytes including the length byte

84 — MPP is set to 132 characters per line

01 — LM is set to column 1

84 — RM is set to column 132

05 — T1 is set to column 5

42 — T2 is set to column 66

Set Vertical Format (SVF) — '0x2BC2[L](MPL)(TM)(BM)(T1)(. . .)(Tn)'

[L] — Length of the parameters, including the length byte

MPL — Maximum Presentation Line; defines the lines per page

TM — Top Margin; line number of top most print position

BM — Bottom Margin; line number of the bottom most print position

T1 — Vertical Tab Stop; line number for a tab stop

Tn — Additional Tab Stops, which can be added in any order

Example

2BC20642053D0A21

2BC2 — SVF

06 — length is 6 bytes

42 — MPL is set to 66 lines per page

05 — TM is set to line 5

3D — BM is set to line 61

0A — T1 is set to line 10

21 — T2 is set to line 33

Set Line Density (SLD) — '0x2BC6[L](Point)'

[L] — length, including length byte. Value of '0x01' denotes default.

Point — Distance to be moved vertically for a single line. The number is indicated in typographic points (one point is equal to 1/72 inch). Setting a value of '0x0C' will result in 6 lines per inch, a value of '0x09' will result in 8 lines per inch. Value of '0x00' denotes default a value of 6 lines per inch.

Example

2BC6020C

2BC6 — SLD

02 — length is 2

0C — 12 points or 6 lines per inch

Set Print Density (SPD) '0x2BD2[L]29(CharDensity)(Resv)'

[L] — length, including length byte. Value of 0x02 denotes default characters per inch (cpi) of 10

CharDensity — value indicating the numbers of cpi

Resv — Reserved (not used)

Example

2BD204290A00

2BD2 — SPD

04 — length is 4 bytes

29 — type

0A — 10 cpi

00 — reserved

Transparent (TRN) — '0x35[L](P1)(. . .)(Pn)'

This SCS control code indicates a section of data that is not scanned for SCS codes, but passed to the print output untouched. The extent of the section of data is denoted by the length byte.

Example

35051B28313055

35 — TRN

05 — length of transparent section, not including the length byte

1B28313055 — transparent section, HP PCL code for PC-8 symbol set 'Esc(10U'

Note
In this example, the transparent section is in ASCII. This would require that the "Transparent is ASCII" box be selected in the Print Services printer session properties in the SNA Manager.

Sample Host Data

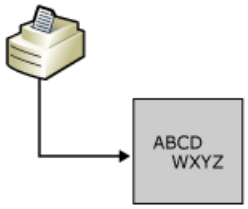
Following is sample data from a host along with an explanation of the data and resulting printout.

```
35021B45 2BC10684 01840542 2BC20642
04420A21 C1C2C3C4 15404040 E6E7E8E9
```

3270 LU 1 Sample Data

Data	Interpretation
35021B45	Transparent section, send 'Esc E', a Reset in HP PCL, to printer
2BC1068401840542	SHF, 132 characters per line, LM 1, RM 132
2BC2064204420A21	SVF, 66 lines per page, TM 4, BM 66
C1C2C3C4	EBCDIC hex values for ABCD
15	New line
404040	EBCDIC hex values for three spaces
E6E7E8E9	EBCDIC hex values for WXYZ

Print output from data in the preceding table. ABCD on the top print line and WXYZ indented on the lower line.



See Also

Other Resources

Host Print Service

Configuring Host Print Service

Configuring Host Print service involves the following steps:

- Creating Link Services
- Creating Connections
- Creating a 3270 Printer LU
- Configuring Host Print Service

In addition, a print demonstration using a script is available. The following procedures will create and configure the print demonstration:

Configuring Host Print service for a 3270 connection

1. In SNA Manager, expand the server on which you want to add print services.
2. Right-click **Print Service**, point to **New**, and then click **3270 Session**.
3. The **Properties** page for this session appears.
4. On the **General** tab, type a session name.
5. Click **Printer**, and then configure a printer.
6. Click the **3270** tab, choose an LU Name to use for this print session. If there are no LU names in the drop-down list, you will need to insert a printer LU on the 3270 connection to be used for this print session (To create a print LU, see the earlier procedure).
7. Configure other parameters as desired.
8. Click **OK** to add this print session.
9. In the console tree, right-click **Print Service**, and then click **Save Configuration**.
10. Right-click **Print Service** again, and then click **Start**.

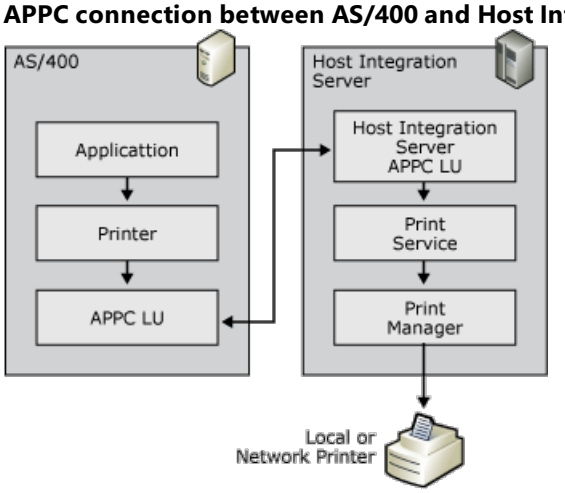
The Host Print service is configured with startup set to Manual. To start the Host Print service at system startup, go to the **Services** icon of Control Panel. Select the service name. Click **Startup**. Change the service activation type from Manual (the default) to Automatic for this service.

SNA Manager will only lock the configuration file when you initiate a configuration change. If the lock is obtained, the status bar will flash **'CONFIG LOCK'**. When you complete the change and save the configuration file, the lock will be released and the status bar will be cleared. The status bar will display **'OUT OF DATE'** on other servers in the domain. To refresh the status on the **'OUT OF DATE'** servers, SNA Manager must be closed and reopened.

AS/400 (APPC) Printing

APPC printing, like LU 1 printing, uses SCS control codes in the data stream. The set of SCS control codes available for use in APPC printing is more extensive, and allows more formatting options than the set described for LU 1 printing. The IBM AS/400 also provides an additional method for formatting print jobs called Host Print Transform (HPT). With HPT enabled, the AS/400 takes responsibility for rendering the print job into data the printer can understand.

APPC connection between AS/400 and Host Integration Server to send print job to local printer



Host Print Transform (HPT)

When SCS control codes are used by the host to format the print output, a print emulator is responsible for translating the SCS codes and characters into data that the printer can understand, through the Windows printer driver and Windows Print system. With HPT enabled, the AS/400 converts the data to printer control codes before sending the data to Host Integration Server. This output from the host requires no further processing after it leaves the AS/400. The print emulator's only responsibility is submitting the data to the printer.

HPT is enabled on the AS/400 in the Device description for the print session. When HPT is enabled, pre-rendered print jobs are sent to the Host Integration Server in marked ASCII Transparent (ATRN) sections using the SCS control code '0x03.' The ATRN control code provides the same function as the Transparent (TRN) control code detailed in the LU 1 printing section. In addition to indicating that the block of data that should be dealt with as transparent, ATRN also indicates that the data is ASCII; therefore it is not converted from EBCDIC to ASCII.

To enable the host transform feature using the default 5224 print device

1. Stop the print writer associated with the print device.
2. Vary off the print device.
3. Issue the following command:

   ```
   chgdevprt devd(<print device>) transform(*YES) mfrtypmdl(<LAN printer type>)
   ```

   Common LAN printer types include: *HP4, *HPIII, *HPII, *IBM4039. To see a complete list of available options, prompt (F4) on the MFRTYPMDL parameter.
4. Vary on the print device.
5. Start the print writer.

For more details on the Host Print Transform feature, see the "OS/400 Printer device programming" manual (SC41-3713), or the "AS/400 Printing IV" redbook (GG24-4389). Both are available from IBM.

SCS Codes

The SCS control codes are fully documented in the *IBM Host Print Guide* (document number SC31-7145). All of the SCS control codes fall within the range of '0x00'–'0x3F.' These codes range from single-byte codes, such as Subscript '0x38' to multiple-byte codes followed by several parameters, such as Set Horizontal Format '0x2BC1...'

The following are some of the more common SCS control codes used.

Note
[L]=length and **(Abv)** represents one-byte parameters in the SCS control code

ASCII Transparency (ATRN) — '0x03[L](P1)(. . .)(Pn)'

This SCS control code indicates a section of data that is not scanned for SCS codes, but passed to the print output untouched. In addition, this control code indicates that the data is ASCII. The extent of the section of data is denoted by the length byte. With HPT jobs, the length byte will commonly be '0xFF'.

Example

030441424344

03 — ATRN

04 — length of 4

41424344 — ASCII hex values for ABCD

SCS Control Code Formatted (Non-HPT)

If HPT is disabled in the Device description for the print session, SCS control codes will be used for the formatting of the print job. The SCS control codes, SHF, SVF, SLD, and SPD detailed earlier for LU 1 printing are also supported in APPC printing. Also commonly used in APPC printing is the SCS control code Presentation Position (PP) '0x34.' This control code allows the print position to be moved either horizontally or vertically, relative to the previous position or to an absolute position.

The following are the four forms of the Presentation Position SCS control code.

Absolute Horizontal Presentation Position (AHPP) '0x34C0(nn)'

nn — column number the print position is set to.

Example

34C00F

34C0 — AHPP

0F — column number 15

Relative Horizontal Presentation Position (RHPP) '0x34C8(nn)'

nn — number of columns to move from the current print position.

Example

34C80F

34C8 — RHPP

0F — 15 columns

Absolute Vertical Presentation Position (AVPP) '0x34C4(nn)'

nn — line number the print position is set to.

Example

34C40F

34C4 — AVPP

0F — line number 15

Relative Vertical Presentation Position (RVPP) '0x344C(nn)'

nn — number of lines to move from the current print position.

Example

344C0F

344C — RVPP

0F — 15 lines

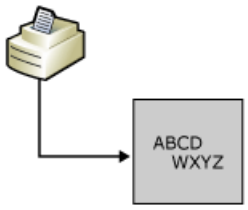
Sample Host Data

Following is sample data from a host along with an explanation of the data and resulting printout.

```
2BC10684 01840542 2BC20642 04420A21
34C404 C1C2C3C4 344C01 34C004 E6E7E8E9
```

Data	Interpretation
2BC1068401840542	SHF, 132 characters per line, LM 1, RM 132
2BC2064204420A21	SVF, 66 lines per page, TM 4, BM 66
34C404	AVPP sets print position to line 4
C1C2C3C4	EBCDIC hex values for ABCD
344C01	RVPP sets print position down one line
34C004	AHPP sets print position to column 4
E6E7E8E9	EBCDIC hex values for WXYZ

Print output from data in preceding table. ABCD on top print line and WXYZ indented on lower line.



See Also

Other Resources

Host Print Service

How to Configure Host Print Service for an AS/400 Computer

Configuring AS/400 print service involves the following steps:

- Creating Link Services
- Creating Connections
- Creating LUs

To create a print LU

1. In SNA Manager, expand **SNA Service** for the server that you are working with, and then expand **Connections**.
2. Right-click the appropriate connection, point to **New**, and then click **Printer LU**.
3. Define your printer LU. You can keep the default for the **LU Number** or assign your own. Although the LU number is meaningful to the connection itself, the number is not very useful, so you can add a user-friendly name in the **LU Name** box.
4. Click **OK**.

To configure Host Print service for an AS/400 Connection

1. In SNA Manager, expand the server on which you want to add print services.
2. Right-click **Print Service**, point to **New**, and then click **APPC Session**.
3. The **Properties** page for this session appears.
4. On the **General** tab, type a session name, click **Printer**, and then configure a printer.
5. On the **APPC** tab, provide an **AS/400 Device Name**, a **Local LU Alias**, a **Mode Name**, and a **Remote LU** to use for this print session.
6. If the **AS/400 Device Name** you specify does not exist on the AS/400, it will be created.
7. On the **Security** tab, configure security for this print session.
8. Configure other parameters as desired.
9. Click **OK** to add this print session.
10. In the console tree, right-click **Print Service**, and then click **Save Configuration**.
11. Right-click **Print Service** again, and then click **Start**.

The Print Service feature of Host Integration Server supports print jobs from an AS/400 that contain basic formatting options. Some print jobs require special formatting. To properly print these jobs, the Host Print Transform feature of the AS/400 must be enabled. Host Print Transform is a feature of the AS/400 operating system (V3R1 and later) that translates print jobs in the SNA character string (SCS) data stream into an ASCII data stream (such as PCL). The data stream is specific to a particular make and model of ASCII printer, which must be defined at the time Host Print Transform is enabled.

Custom Host Code Page

Host Integration Server allows a custom host code page to be used for a printer session. The host code page is used for the translation between ASCII and EBCDIC. By default, a printer session will use the standard language code page provided by Windows Server 2003 or Windows 2000. As an alternative, a custom code page can be specified to allow a different translation. For example, using the default code page, the EBCDIC letter "A" ('0xC1') would be translated to an ASCII letter "A" ('0x41'). With a custom code page, it would be possible to have the EBCDIC "A" translated to any value. The custom code pages are text files that can be modified with a hex editor. Samples are included in the PRINTSERVERADDONS directory on the Host Integration Server CD-ROM. The code page file contains 512 bytes. The first 256 bytes represent what the EBCDIC characters are translated to. The second 256 bytes are what the ASCII characters are translated to. Logically each section is a 16-column by 16-row block.

Bytes 0-255: Data from Host

```
   | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
10| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
30| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
40| 20 a0 e2 e4 e0 e1 e3 e5 e7 f1 a2 2e 3c 28 2b 7c
50| 26 e9 ea eb e8 ed ee ef ec df 21 24 2a 29 3b ac
60| 2d 2f c2 c4 c0 c1 c3 c5 c7 d1 a6 2c 25 5f 3e 3f
70| f8 c9 ca cb c8 cd ce cf cc 60 3a 23 40 27 3d 22
80| d8 61 62 63 64 65 66 67 68 69 ab bb f0 fd de b1
90| b0 6a 6b 6c 6d 6e 6f 70 71 72 aa ba e6 b8 c6 a4
a0| b5 7e 73 74 75 76 77 78 79 7a a1 bf d0 dd fe ae
b0| 5e a3 a5 b7 a9 a7 b6 bc bd be 5b 5d af a8 b4 d7
c0| 7b 41 42 43 44 45 46 47 48 49 ad f4 f6 f2 f3 f5
d0| 7d 4a 4b 4c 4d 4e 4f 50 51 52 b9 fb fc f9 fa ff
e0| 5c f7 53 54 55 56 57 58 59 5a b2 d4 d6 d2 d3 d5
f0| 30 31 32 33 34 35 36 37 38 39 b3 db dc d9 da 00
```

Bytes 256-511: Data to Host

```
   | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00| 00 01 02 03 37 2d 2e 2f 16 05 25 0b 0c 0d 0e 0f
10| 10 14 24 04 b6 15 32 26 18 19 00 27 1c 1d 1e 1f
20| 40 5a 7f 7b 5b 6c 50 7d 4d 5d 5c 4e 6b 60 4b 61
30| f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 7a 5e 4c 7e 6e 6f
40| 7c c1 c2 c3 c4 c5 c6 c7 c8 c9 d1 d2 d3 d4 d5 d6
50| d7 d8 d9 e2 e3 e4 e5 e6 e7 e8 e9 ba e0 bb b0 6d
60| 79 81 82 83 84 85 86 87 88 89 91 92 93 94 95 96
70| 97 98 99 a2 a3 a4 a5 a6 a7 a8 a9 c0 4f d0 a1 00
80| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0| 41 aa 4a b1 9f b2 6a b5 bd b4 9a 8a 5f ca af bc
b0| 90 8f ea fa be a0 b6 b3 9d da 9b 8b b7 b8 b9 ab
c0| 64 65 62 66 63 67 9e 68 74 71 72 73 78 75 76 77
d0| ac 69 ed ee eb ef ec bf 80 fd fe fb fc ad 8e 59
e0| 44 45 42 46 43 47 9c 48 54 51 52 53 58 55 56 57
f0| 8c 49 cd ce cb cf cc e1 70 dd de db dc 8d ae df
```

The value being translated is matched to its new value using the first number in the hex value as the row and the second as the column. For example, to find what the EBCDIC character "Z" ('0xE9') is translated to in the sample code page count down to row E and over to column 9. This position has the value '0x5A', which is the ASCII value for a "Z".

Sample Host Code Page (as seen in a Hex editor)

```
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
```

```
20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
20 A0 E2 E4 E0 E1 E3 E5-E7 F1 A2 2E 3C 28 2B 7C
26 E9 EA EB E8 ED EE EF-EC DF 21 24 2A 29 3B AC
2D 2F C2 C4 C0 C1 C3 C5-C7 D1 A6 2C 25 5F 3E 3F
F8 C9 CA CB C8 CD CE CF-CC 60 3A 23 40 27 3D 22
D8 61 62 63 64 65 66 67-68 69 AB BB F0 FD DE B1
B0 6A 6B 6C 6D 6E 6F 70-71 72 AA BA E6 B8 C6 A4
B5 7E 73 74 75 76 77 78-79 7A A1 BF D0 DD FE AE
5E A3 A5 B7 A9 A7 B6 BC-BD BE 5B 5D AF A8 B4 D7
7B 41 42 43 44 45 46 47-48 49 AD F4 F6 F2 F3 F5
7D 4A 4B 4C 4D 4E 4F 50-51 52 B9 FB FC F9 FA FF
5C F7 53 54 55 56 57 58-59 5A B2 D4 D6 D2 D3 D5
30 31 32 33 34 35 36 37-38 39 B3 DB DC D9 DA 00
00 01 02 03 37 2D 2E 2F-16 05 25 0B 0C 0D 0E 0F
10 14 24 04 B6 15 32 26-18 19 00 27 1C 1D 1E 1F
40 5A 7F 7B 5B 6C 50 7D-4D 5D 5C 4E 6B 60 4B 61
F0 F1 F2 F3 F4 F5 F6 F7-F8 F9 7A 5E 4C 7E 6E 6F
7C C1 C2 C3 C4 C5 C6 C7-C8 C9 D1 D2 D3 D4 D5 D6
D7 D8 D9 E2 E3 E4 E5 E6-E7 E8 E9 BA E0 BB B0 6D
79 81 82 83 84 85 86 87-88 89 91 92 93 94 95 96
97 98 99 A2 A3 A4 A5 A6-A7 A8 A9 C0 4F D0 A1 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
41 AA 4A B1 9F B2 6A B5-BD B4 9A 8A 5F CA AF BC
90 8F EA FA BE A0 B6 B3-9D DA 9B 8B B7 B8 B9 AB
64 65 62 66 63 67 9E 68-74 71 72 73 78 75 76 77
AC 69 ED EE EB EF EC BF-80 FD FE FB FC AD 8E 59
44 45 42 46 43 47 9C 48-54 51 52 53 58 55 56 57
8C 49 CD CE CB CF CC E1-70 DD DE DB DC 8D AE DF
```

See Also

Concepts

AS/400 (APPC) Printing

Formatting Print Jobs

Host Print service can format print jobs by either using the Windows Server 2003 or Windows 2000 printer driver or by using a Printer Definition Table.

By default, print jobs submitted by Host Print service to a Windows Server 2003 or Windows 2000 Server print queue rely on the Windows Server 2003 or Windows 2000 printer driver to format the print data and send to the physical printer. Formatting the print data is done by the Windows Graphical Device Interface (GDI).

In addition to the Windows GDI, Host Integration Server Host Print service supports the use of a Printer Definition Table (PDT), which bypasses the formatting function of the Windows Server 2003 or Windows 2000 printer driver. The PDT file provides a function similar to a printer driver, in that it translates Graphic Device Interface (GDI) calls to control codes specific for a printer.

Selecting PDT causes the Windows Server 2003 or Windows 2000 Host Print service to treat all received data as transparent. All data is passed directly to the printer, except SCS codes, which are treated differently. Using a PDT, the SCS codes will be translated using the PDT before they are passed to the printer. For more information about Printer Definition Files, see Printer Definition Files.

The PDT is created in two steps.

1. A source text file is created and called the Printer Definition File (PDF) that defines the codes that can be used to control the printer.

2. A program is used to compile the information in the PDF into a binary file, the PDT that is used by Host Print service.

For example, if the host sends a byte indicating a new line ('0x15'), the PDT could be used to convert this to a carriage return, line feed ('0x0D0A').

For more information about creating a PDT, see Printer Definition Files.

To enable GDI

1. In SNA Manager, expand the server, and then click **Print Service**.
2. Right-click a print service displayed in the details pane, and then click **Properties**.
3. Click the **Job Format** tab, and then select **GDI**.

To select a PDT for printing

1. In SNA Manager, expand the server, and then click **Print Service**.
2. Right-click a print service displayed in the details pane, and then click **Properties**.
3. Click the **Job Format** tab, select **PDT**, and then click **PDT File**. The **Select Compiled Printer Definition File** dialog box appears.
4. Select the PDT that you want to use, and then click **Open**.
5. Click **OK**.

Transparency

Transparency sets a flag that indicates that transparent data from the host is in ASCII and needs no translation from EBCDIC to ASCII. Selecting **Transparency is ASCII** causes the Windows Server 2003 or Windows 2000 Host Print service not to put the received data through an EBCDIC to ASCII translation table before printing.

Check **Transparency Custom Byte** to send the data stream in transparent mode. **Transparency Custom Byte** indicates the character designated to start a sequence of transparent data (the transparent data may or may not be ASCII). The IBM standard is 0x35, but if the host print job uses another value (for example, 0x36), this should be specified.

To select Transparency

1. In SNA Manager, expand the server, right-click **Print Service**, and then click **Properties**.
2. Click the **Advanced** tab, click **Transparency Custom Byte**,enter the byte data, and then click **OK**.

See Also
Tasks
Formatting Print Jobs

Printer Definition Files

Host Print service enables you to specify the capabilities of a printer to override the defaults provided by the Windows printer driver.

By default, jobs submitted by Host Print service to a Windows print queue rely on the printer's Windows driver to send to the physical printer the data required to perform such formatting tasks as breaking lines and starting a new page.

Applications that bypass the formatting function of the Windows printer driver can use codes specified in a printer definition file to control the physical printer for a particular printer session under Host Print service. To use the file, open the properties page of the printer session, click the **Printing** tab, select **PDT**, and then type the fully qualified path of the file in the box. For more information about using the PDT file, click **Help** on the printer session properties dialog box.

Creating a printer definition file consists of two steps: First, you create a source text file that defines the codes that can be used to control the printer. Second, you run a program to compile the information in the text file into a binary file that can be used by Host Print service.

In This Section

Creating the Source Text File

You can use any text editor (such as Notepad) to create the source text file. The conventional file extension for the source text file is .pdf (which stands for Printer Definition File).

The PDF file consists of two sections:

- The Macro Definition Section
- The Parameter Definition Section

All text bounded by the C-language comment markers (**/*** and ***/**) is treated as comments and ignored by the compiler.

The next two sections describe the PDF sections currently supported by Host Print service.

Macro Definition Section

The macro definition section is bounded by **BEGIN_MACROS** and **END_MACROS** statements and consists of lines with the following syntax.

macro_name **EQU** *control_character_list*

The macro name can be any alphanumeric string that does not contain spaces, and the control character list consists of one or more two-digit hexadecimal values representing the data to be sent to the printer.

For example, the following macro defines the NUL character.

```
NUL EQU 00
```

For an example macro definition, see Sample Source Text File.

Parameter Definition Section

The parameter definition section immediately follows the macro definition section and consists of one or more lines with the following syntax.

parameter_name **=** *value_list*

The parameter name can be any character string that does not contain spaces. The compiler ignores parameter names not supported by Host Print service.

The value list can be empty or can contain one or more of the following:

- A three-digit decimal value.
- A two-digit hexadecimal value.
- A one-char character value.
- The name of a macro specified in the macro definition section.

For example, the following shows a parameter defining the control sequence to be sent to the printer to begin a new line.

```
NEW_LINE = CRR LFF
```

In this example, CRR and LFF are the names of macros specified in the macro definition section.

Host Print service currently supports the following parameters (definitions of unsupported parameters are ignored).

Parameter name	**Description**
START_JOB	Control sequence to be sent at the start of a print job.
END_JOB	Control sequence to be sent at the end of a print job.
CARRIAGE_RETURN	Control sequence for a carriage return.
LINE_FEED	Control sequence for a line feed.
FORM_FEED	Control sequence for a form feed.
NEW_LINE	Control sequence for a new line.
SET_6_LINES_PER_INCH	Control sequence to specify 6 LPI.
SET_8_LINES_PER_INCH	Control sequence to specify 8 LPI.
START_HIGHLIGHT_INTENSE	Control sequence to begin bold printing.
END_HIGHLIGHT_INTENSE	Control sequence to end bold printing.
START_HIGHLIGHT_UNDERLINE	Control sequence to begin underline printing.
END_HIGHLIGHT_UNDERLINE	Control sequence to end underline printing.
KANJI_CODE?	Control sequence of Kanji code for a printer, either JIS or SHIFT_JIS.
KANJI_ON	Control sequence to start printing Kanji.
KANJI_OFF	Control sequence to end printing Kanji.
SET_PAGE_LENGTH	Control sequence to set number of lines per page.

LEFT_MARGIN	Control sequence to set left margin in number of characters.
RIGHT_MARGIN	Control sequence to set right margin in number of characters.
TOP_MARGIN	Control sequence to set top margin in number of lines.
SET_HORIZONTAL_POSITION	Control sequence to set row position.
SET_VARIABLE_LINE_DENSITY	Control sequence to set line density.
SET_VARIABLE_PRINT_DENSITY	Control sequence to set number of characters per inch.
SET_FONT_SIZE	Control sequence to set font size in points.

See Also

Other Resources

Printer Definition Files

Sample Source Text File

The sample PDF file, HPLJ2.PDF, is provided on the Host Integration Server CD-ROM in the \SDK\Samples\SNA\PrintDefFile directory.

See Also

Other Resources

Printer Definition Files

Compiling the Source Text File

After you have created the source text file, the next step is to compile the file using the Pdfcomp (PDFCOMP.EXE) utility located in the *platform*\SYSTEM\PRINTSRV directory of the Host Integration Server CD-ROM. To run Pdfcomp, type the following at the command prompt.

```
input_file output_file
```

You must specify the full name (including file extensions) of the input and output files.

For example, to compile the sample PDF file, copy PDFCOMP.EXE and HPLJ2.PDF to a directory on the server's hard disk, open a Command Prompt window and change the current directory to the directory containing the copied files. Then, type the following at the command prompt.

```
hplj2.pdf hplj2.pdt
```

This creates the binary PDT file in the current directory. Specifying this file in the properties of a printer session allows Host Print service to send preformatted data to the printer, bypassing the formatting function of the Windows printer driver.

Compilation of the PDF file into a PDT is not strictly necessary. If an uncompiled PDF file is selected in a Print Session under the SNA Manager, the compilation is performed automatically each time the print session is used. This feature has a performance overhead and is available mainly for ease of development. For production systems, it is strongly recommended that you pre-compile PDF files.

See Also

Other Resources

Printer Definition Files

TN Service

TN Service, which is based on the Telnet protocol, enables a user at one site to connect and interact with a remote system at another site using the TCP/IP network protocol. When connecting to a remote computer, the initiating computer acts as a terminal for the remote computer. Host Integration Server supports TN3270-style and TN5250-style sessions in addition to native Host Integration Server client 3270 and 5250 sessions.

In This Section

TN3270

TN3270 service utilizes the features of Host Integration Server to obtain mainframe access and to address issues such as security and redundancy. TN3270 service supports the TN3270, TN3287, and TN3270E protocols, providing terminal emulation and printing capabilities.

To provide for TN3270 service, in Host Integration Server you create connections and LU definitions that map to the mainframe. TN3270 communicates with Host Integration Server using the logical unit application (LUA) API. Therefore, all LUs configured for use with the TN3270 service must be LUA LUs. The LUA LUs and LUA pools defined for the Host Integration Server computers can then be assigned to the TN3270 service using the drag-and-drop method. When the system is activated, the LUA LUs become available for TN3270 clients to access mainframe applications.

Redundancy involves SNA local nodes and link services. Each Microsoft Windows Server 2003 or Windows 2000 domain can contain one or more SNA subdomains. LUA LUs from multiple servers can be assigned to the TN3270 service. This results in client sessions distributed among the participating servers in the subdomain, balancing the load. This also allows redundancy between Host Integration Server computers. If one server goes down, a client computer can then access LUA LUs on a different server.

Similarly, a server can be configured with redundant host links to increase fault tolerance and bandwidth.

In This Section

IP Settings

The IP settings assigned to an LUA LU or pool will allow TN3270 clients to connect to that LUA LU or pool. By default, the LUA LU or pool is not assigned an IP address or a subnet mask. This will allow any TN3270 client to use the LUA LUs or pools.

If you want to restrict an LUA LU or pool to one or more specific TN3270 clients, you should determine the IP address or host name of the client computers that will use these LUA LUs or pools. For client computers using Windows Server 2003 or Windows 2000, you can determine the IP address, subnet mask, and host name by typing **ipconfig/all** from the command prompt on the client computer.

If a workstation has a name that can be resolved using name resolution, it can be used in place of IP addresses. For example, if a workstation name is Giraffe and if the IP name list contains that name, it will map to the correct IP address. Name resolution works with WINS, DHCP, or similar name resolution services. For more information regarding WINS and DHCP, see your Windows Server 2003 or Windows 2000 documentation.

You can modify, delete, or add IP addresses and subnet masks within LUA LUs or pools. If you want to change the configuration of multiple LUA LUs, you can only change properties such as display types, the IP address, and subnet mask that the LUA LUs have in common. You cannot change properties such as the LUA LU name or number, because these values are unique for each LUA LU.

You can add new IP addresses and subnet masks to a range of LUA LUs. If the new IP address or subnet mask already exists on some of the LUA LUs, but not on others, the addition will occur without duplication in the ones that already have the IP address or subnet mask. You can also modify or delete the IP addresses that the LUA LUs have in common and that appear on the IP address list.

Configuration changes are apparent only to users who establish a connection after the changes are saved. Users who were connected at the time the configuration changes were made will not be affected.

See Also

Other Resources

TN3270

Administering TN3270

Using SNA Manager, you can view client addresses, LU names, and the state of each defined client computer. The session status information is useful if you need to stop, pause, or modify TN3270.

You can monitor network activity, including the state of each LU. An LU or pool will display one of the following states.

State	Description
CONNECTED	The TN client has established a connection to TN3270 service.
SSCP-LU	The TN client has established a connection to VTAM.
LU-LU	The LU is bound to a host application.
TERM	The connected session is terminating.

To add LUs or pools to TN3270 service

1. Before LUs can be added to the TN3270 service, you must first create application (LUA) LUs on an appropriate connection.
2. 0Select the LUs on the connection.

   A contiguous range of LUs can be selected by using the mouse to select the first item in the range you want to add, holding down the SHIFT key, and using the mouse to select the last item in the range that you want to add.

   A noncontiguous range of LUs can be selected by using the mouse to select each item in the range that you want to add, while holding down the CTRL key.
3. Drag the LUs into the desired server icon in the TN3270 Service folder.

   The LU appears in the list frame when the server icon in the TN3270 Service folder is highlighted. The icon for the LUs will change to a TN icon both in the TN3270 service and in the connection list.

   By default, the LU will be assigned an IP address of 0.0.0.0 and a subnet mask of 0.0.0.0. This will allow any TN3270 client to use this LU.
4. Specify one or more client IP addresses for this LU.
5. On the **Action** menu, click **Save Configuration** to put the changes into effect.

By adding an LU, you are defining Host Integration Server resource access to TN3270 service and you are defining logmode entries that are associated with a Host Integration Server resource.

Configuration changes are apparent only to users who establish a connection after the configuration changes are saved. Users who were connected at the time that the configuration changes were made will not be affected.

You can modify, delete, or add IP addresses and subnet masks to LUA LUs. If you want to change the configuration of multiple LUA LUs, you can only change properties such as display types, the IP address, and subnet mask that the LUA LUs have in common. You cannot change properties such as the LUA LU name or number, because these values are unique for each LUA LU.

You can add new IP addresses and subnet masks to a range of LUA LUs. If the new IP address or subnet mask already exists on some of the LUA LUs, but not on others, the addition will occur without duplication in the ones that already have the IP address or subnet mask. You can also modify or delete the IP addresses that the LUA LUs have in common and that appear on the IP address list.

You can assign a 3270 LU pool to a workstation, not an LUA pool. LUA pools can be assigned to the TN3270 service.

To edit a TN3270 LU configuration

1. In the **SNA Manager** console tree, select the LU that you want to view or modify.
2. Right-click the LU name, and then click **Properties**.

3. Click **OK** to exit.

TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client computer to connect to multiple host computers.

To override the default port value for a session

1. Select an LU.
2. Right click, and then click **Properties**. The **TN3270 LU Properties** dialog box appears.
3. Click **Use**, and then type a port number other than the default used for the server. The LU port assignment will override the default port assigned to the server.

To start, pause, continue, and stop TN3270 service

1. Right-click **TN3270**, and then click **Start** or **Stop**.

   - or -

2. In the **Services** utility of the Windows Server 2003 or Windows 2000 **Administrative Tools or Windows Control Panel**, select **TN3270 Service**, and right-click **Start**, **Pause**, **Continue**, or **Stop**.

The TN3270 service is set to start manually by default. You can change this to automatic if you are not running either the TN5250 service or the Telnet daemon on this server, or if you have configured the TCP ports for more than one of these services.

Once TN3270 service has stopped, it can no longer be accessed. You may need to start the TN3270 service after you have paused or stopped it. TN3270 service can be restarted only on the local system.

Pausing allows you to prevent new users from establishing a connection with TN3270 service without disconnecting current users. You can then view TN3270 service session status and notify connected users to disconnect from TN3270 service.

Before stopping TN3270 service, notify all connected users that they will be disconnected within a specified time period. Stop the service after expiration of your warning period.

Tips

- To start TN3270 service from a command prompt, type

  net start tn3270

- To pause TN3270 service from a command prompt, type

  net pause tn3270

- To continue TN3270 service from a command prompt, type

  net continue tn3270

- To stop TN3270 service from a command prompt, type

  net stop tn3270

TN3270, TN5250, and Telnet services all default to the well-known TCP port number 23. If you plan to install more than one of these services, perform the following steps.

To use TN3270 service with TN5250 service or Telnet service

1. Configure the services to use unique port numbers.
2. Reconfigure all your TN3270 clients or TN5250 clients to use the new port numbers.

To remove TN3270 service

1. To pause TN3270 service, use the **Services** utility in Windows Server 2003 or Windows 2000 Control Panel.
2. Pausing TN3270 service allows you to notify connected users to disconnect from TN3270 service before you stop and remove the application.
3. To stop TN3270 service, use the **Services** utility in Windows Server 2003 or Windows 2000 Control Panel.
4. Open **Control Panel**, and double-click **Add/Remove Programs**.
5. Click **Host Integration Server**, and then click **Change**. The **Add/Remove Application** dialog box appears.
6. In the **Add/Remove** dialog box, click **Add/Remove**.
7. Click the **TM3270 Service** icon, and then click **Entire Feature will be unavailable**.
8. Click **Continue**.

Note
You can remove TN3270 service anytime you want. However, removing TN3270 service deletes the TN3270 service files from your computer, including TN3270 service configuration data. To use TN3270 service again, you must run Host Integration Server Setup to reinstall TN3270 service files.

See Also
Concepts
TN3270 and Single Sign-On

TN3270 and Single Sign-On

The TN3270 service does not work with Single Sign-On. You should not enable the TN3270 service to run under a user account for which you have a Single Sign-On mapping. If you access the host through the TN3270 service by typing **MS$SAME** for your logon, you will get the user ID and password of the user under which the TN3270 service is running (for example, SNAUSER). If you use an Administrator-level account and change the password, SNA services will fail to start after the password change.

TN3270 Configuration

This section describes the management and configuration of certificates necessary for running TN3270 on Host Integration Server.

In This Section

Managing Certificates

Configuring Certificates

Managing Certificates

The SChannel API uses certificates to provide its security features.

In This Section

Server Authentication

Client Authentication

Obtaining and Creating Certificates

Server Authentication

For server authentication, the server requires a valid certificate with the following properties:

- Type X509
- Suitable for Server Authentication
- Associated private key
- Stored in the Personal or My certificate store for the service account used by the TN3270 service
- By default, a TN3270 server will look for a certificate with a Common Name (CN) matching the host name of the computer running the TN3270 server. This default can be changed by using a registry entry. For details, see Changing the Default Server Authentication Certificate Common Name (CN).

This certificate will be sent to the client as part of the handshake negotiation when the connection is established. For the client to accept the certificate:

- The certificate (and its issuing chain) must be current (for example, not outside of its valid dates).
- The issuing chain must lead to a certification authority (CA) that appears in the clients Trusted Root CA List.
- The certificate (or any part of its issuing chain) should not appear on a certificate revocation list (CRL) of its issuer.
- Most clients offer strict certificate checking, which if selected, will reject connections if the server certificates common name does not match its host name.

Note
If the certificate on the server is changed, the TN3270 server must be stopped and restarted.

Client Authentication

For client authentication, the client requires a valid certificate with the following properties:

- Type X509
- Suitable for Client Authentication
- Associated private key

You might not want to grant access to some of these certificate settings. It is recommended that you check the list of default Trusted Root Certification Authorities on the server, and remove any you do not want.

How the certificate is stored and selected depends on the client program. The certificate will be passed to the server as part of the handshake process. For the server to accept the certificate:

- The certificate (and its issuing chain) must be current.
- The issuing chain must lead to a CA that appears in the servers Trusted Root CA list.
- The certificate (or any part of its issuing chain) should not appear on a CRL of its issuer.

See Also
Concepts
Server Authentication
Obtaining and Creating Certificates

Obtaining and Creating Certificates

It is recommended that you use the Microsoft Certificate Services and certification authorities to manage certificates.

When a certificate is issued, it includes a certificate and a private key. When the certificate is transmitted for verification purposes, only the certificate part is sent (and not the private key). The server needs a certificate and a private key for the server authentication certificate. The server needs a copy of the client authentication certificate for its root CA.

Obtaining Certificates

Certificates are obtained from certification authorities. Because these CAs are widely trusted organizations, the certificate will be recognized widely.

Creating Certificates

Windows Server 2003 and Windows 2000 Server feature a certification authority program that allows a local CA to be set up. The local CA may depend on a certificate obtained from an external, well-trusted CA, or it may be a stand-alone CA.

This system can be used if the root certificate (on the CA program) is copied to the Trusted Root CAs store on all the computers using the certificates. The local CA can then issue client and server authentication certificates, and each will be recognized by the other.

See Also

Concepts

Server Authentication

Client Authentication

Configuring Certificates

The **TN3270 Server properties** dialog box contains a **Port/Security** tab, which allows you to configure the ports to be available to the TN3270 server. The security parameters are then configured on a port-by-port basis.

The following issues are required to implement security support:

- A client that attempts to connect to a port that has not been configured on the TN3270 server will be unsuccessful.
- The default port defined in SNA Manager does not affect the security configuration. Either a port security record is found (in which case it is used), or it is not found (in which case the client fails to connect).
- A port with security level Unsecured means that TLS/SSL will be fully disabled on that port. There will be no exchange of certificates on that port.
- All the changes made to the configuration are dynamic.
- Server authentication certificates may not be dynamically changed.

In This Section

Switching on Security and Changing Certificates

To support security, the TN server needs to load a server authentication certificate. This is done when a TN server receives a configuration requiring security (for example, at least one port is configured to a security level other than Unsecure) for the first time. After a certificate has been loaded successfully, the certificate cannot be changed without restarting the TN server.

If the certificate-loading process fails (for example, if the certificate is not found or is invalid), any ports requiring security will not be available to the TN server and an error will be logged. The user must then fix the certificate and try again.

Note
The certificate-loading process incorporates several stages, such as opening a certificate store, finding the certificate, and acquiring credentials based on the certificate. Obtaining credentials involves three steps:

- Opening the servers certificate store (using **CertOpenStore**)
- Obtaining the server authentication certificate (using **CertFindCertificateInStore**)
- Getting a credential for each security setting (using **AcquireCredentialHandle**)

The credential contains all the security options supported by the credential (such as maximum and minimum encryption strength, and algorithms supported). Client authentication is not a credential property. The credential is linked to the server authentication certificate.

If security is specified but the TN3270 server fails to get the credentials, any ports that are defined as secure will be unavailable to clients. An error will be logged and the user can try to reload the credentials again.

See Also

Concepts

Changing the Default Values of the Security Parameters

Changing the Default Server Authentication Certificate Common Name (CN)

Other Resources

Configuring Certificates

Changing the Default Values of the Security Parameters

By default, the security levels are:

- High = 168-bit encryption (minimum)
- Medium = 128-bit encryption (minimum)
- Low = 40-bit encryption (minimum)
- Unsecured = TLS/SSL fully disabled

The default values of the first three of these levels can be overridden by the following registry entries (stored in **HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/TN3270/Parameters**):

- **SSLHighSecurity**
- **SSLMediumSecurity**
- **SSLLowSecurity**

Each registry entry will contain a numeric (DWORD) value. The registry is checked for entries only when the TN3270 server is started. For any changes in the registry entries to take effect, the TN3270 server must be restarted.

See Also
Concepts
Switching on Security and Changing Certificates
Changing the Default Server Authentication Certificate Common Name (CN)
Other Resources
Configuring Certificates

Changing the Default Server Authentication Certificate Common Name (CN)

By default, the TN3270 server will look for a certificate with a common name that matches its host name, for example, the name returned by **gethostname**. This can be changed by the following registry entry (stored in **HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/TN3270/Parameters**):

- **SSLServerCertCN**

This entry contains a string containing the new CN for the certificate. The registry is checked for entries only when the TN3270 server is started. For any changes in the registry entries to take effect, the TN3270 server must be restarted.

See Also
Concepts
Switching on Security and Changing Certificates
Changing the Default Values of the Security Parameters
Other Resources
Configuring Certificates

TN5250

TN5250 acts as a gateway that allows an AS/400 access to TN5250 clients providing 5250 terminal emulation.

To provide for TN5250 service, you must create TN5250/AS400 definitions that map to the AS/400. TN5250/AS400 definitions include local and remote APPC LUs, the mode (the default is QPCSUPP), AS/400 user name and password, terminal type, and IP address and subnet mask.

TN5250 supports multiple sessions up to the limits of the mode used in the TN5250/AS400 definition.

IP Settings

IP settings assigned to TN5250/AS400 definitions allow TN5250 clients to connect to the AS/400. By default, the TN5250/AS400 definition is not assigned an IP address or a subnet mask. This will allow any TN5250 client to connect to the AS/400.

If you want to restrict the use of a TN5250/AS400 definition to one or more specific TN5250 clients, you should determine the IP addresses or host names of the client computers that will use the TN5250/AS400 definition. For client computers using Windows Server 2003 or Windows 2000, you can determine the IP address, subnet mask, and host name by typing **ipconfig/all** at the command prompt on the client computer.

If a workstation's name can be resolved using name resolution, it can be used in place of IP addresses. For example, if a workstation name is Giraffe, and if the IP name list contains that name, it will allow that workstation access. Name resolution works with WINS, DHCP, or similar name-resolution services. For more information about WINS and DHCP, see the Windows Server 2003 or Windows 2000 Help.

You can modify, delete, or add IP addresses and subnet masks within a TN5250/AS400 definition. If you want to change the configuration of multiple TN5250/AS400 definitions, you can only change the properties of items that the TN5250/AS400 definitions have in common. You cannot change the name of remote LUs.

You can add new IP addresses and subnet masks to multiple TN5250/AS400 definitions. If the new IP address or subnet mask already exists on some of the TN5250/AS400 definitions, but not on others, the addition will occur without duplication in the ones that already have the IP address or subnet mask. You can also modify or delete the IP addresses that the TN5250/AS400 definitions have in common and that appear on the IP address list.

Configuration changes are apparent only to users who establish a connection after the changes are saved. Users who were connected at the time the configuration changes were made are not affected.

See Also

Tasks

TN5250 Administration

TN5250 Administration

The local LU, remote LU, and mode must match the configuration information in Host Integration Server.

To enable an APPC session with the AS/400, the user ID and password must be provided for conversation security. Contact your AS/400 administrator for the correct information.

The TN5250 requires TN5250/AS400 definition terminal names to allow TN5250 service to accept client requests from client computers emulating those types of terminals.

To set up Host Integration Server for TN5250 access

1. Open **SNA Manager** console tree.
2. Install and configure a link service, if this has not been done.
3. Insert a connection to an AS/400 that the TN5250 clients will access. Configure the connection with Remote End as Peer System.
4. Configure a local LU and a remote LU for access to an AS/400, making sure the mode for the remote LU is QPCSUPP.
5. On the **File** menu, click **Save Configuration** to put the changes into effect.

To add and configure LUs for TN5250 service

1. Before LUs can be added to the TN5250 service, you must first install a link service, add a connection to an AS/400 on this link service, and create local and remote LUs for accessing this AS/400.
2. Right-click **TN5250**, point to **New**, and then click **TN5250 AS/400 Definition**.
3. Configure the properties of this TN5250 AS/400 definition.

   If you do not specify an IP address for an LU, the default value will allow any TN5250 client computer access to this LU.

   Click **Help** for information on the property options.
4. Click **OK** to close the **AS/400 Definition Properties** dialog box.
5. On the **Action** menu, click **Save Configuration** to put the changes into effect.

Configuration changes are apparent only to users who establish a connection after the configuration changes are saved. Users who were connected at the time that the configuration changes were made will not be affected.

You can modify, delete, or add IP addresses and subnet masks to AS/400 definitions. If you want to change the configuration of multiple AS/400 definitions, you can only change properties such as display types, the IP address, and subnet mask that the AS/400 definitions have in common. You cannot change properties such as the local or remote LUs, because these values are unique for each AS/400 definition.

You can add new IP addresses and subnet masks to a range of AS/400 definitions. If the new IP address or subnet mask already exists on some of the AS/400 definitions, but not on others, the addition will occur without duplication in the ones that already have the IP address or subnet mask. You can also modify or delete the IP addresses that the AS/400 definitions have in common and that appear on the IP address list.

To edit an AS/400 definition for TN5250 service

1. In the SNA Manager console tree, select the TN5250 service AS/400 definition that you want to view or modify, right-click, and then click **Properties**.

To delete AS/400 definitions from TN5250 service

1. In the SNA Manager console tree, select the icon for the AS/400 definition that you want to delete.
2. Click **Delete**.
3. Click **Save** to save your configuration changes.

Note

Configuration changes are apparent only to users who establish a connection after the configuration changes are saved. Users who were connected at the time that the configuration changes were made will not be affected.

To start, pause, continue, and stop TN5250 service

1. Right-click **TN5250**, and then click **Start** or **Stop**.

   - or -

2. In the **Services** utility of the Windows Server 2003 or Windows 2000 **Administrative Tools**, select **TN5250 Service** and click **Start**, **Pause**, **Continue**, or **Stop**.

The TN5250 service is set to start manually by default. You can change this to automatic if you are not running either the TN5250 service on this server, or if you have configured the TCP ports for more than one of these services.

After TN5250 service has stopped, it can no longer be accessed. You may need to start the TN5250 service after you have paused or stopped it. TN5250 service can be restarted only on the local system.

Pausing allows you to prevent new users from establishing a connection with TN5250 service without disconnecting current users. You can then view session status in the Active TN5250 Sessions folder and notify connected users to disconnect from TN5250 service.

Before stopping TN5250 service, notify all connected users that they will be disconnected within a specified time period. Stop the service after expiration of your warning period.

Tips

- To start TN5250 service from a command prompt, type

  net start tn5250

- To pause TN5250 service from a command prompt, type

  net pause tn5250

- To continue TN5250 service from a command prompt, type

  net continue tn5250

- To stop TN5250 service from a command prompt, type

  net stop tn5250

Pausing or removing TN5250 service

1. To pause TN5250 service, use the **Services** utility in Windows Server 2003 or Windows 2000 Control Panel.

   Pausing TN5250 service allows you to notify connected users to disconnect from TN5250 service before you stop and remove the application.

2. To stop TN5250 service, use the **Services** utility in Windows Server 2003 or Windows 2000 Control Panel.
3. Open **Control Panel**, and then double-click **Add/Remove Programs**.
4. Click **Host Integration Server**, and then click **Change**. The **Add/Remove Programs** dialog box appears.
5. In the **Add/Remove Application** dialog box, click **Add/Remove**.
6. Click the **TN5250 Service** icon, and then click **Entire feature will be unavailable**.
7. Click **Continue**.

Note

You can remove TN5250 service anytime you want to do so. However, removing TN5250 service deletes the TN5250 service files from your computer, including TN5250 service configuration data. To use TN5250 service again, you must run Host Integration Server Setup to reinstall TN5250 service files.

See Also

Concepts

TN5250

Active Directory Services

Microsoft Active Directory is the Microsoft Windows Server 2003 or Windows 2000 directory service.

A directory service is an object-oriented information database of network resources. It also provides the services that locate, use, and manage the database and the network resources. These network resources are known as objects and can include network servers, users, printers, computers, databases, and security policies.

Active Directory provides the following services and benefits:

- **Simplified administration.** Active Directory provides a single point of administration for all network resources. It also provides a single point of logon for network users.

- **Network security.** Security policies can be implemented and enforced to help keep information and resources safe, and at the same time, make information and resources available to the right people.

- **Scalability.** The Active Directory database can be expanded by dividing it into partitions. A partition is a logical division of objects. Using partitions, Active Directory can scale from small installations with a few objects, to large installations containing millions of objects.

- **Directory replication and distribution.** Replicating and distributing the directory to other servers in the network ensures its availability and redundancy.

- **Open standards.** Active Directory supports industry standard naming conventions, such as the Domain Name System (DNS), and protocols, such as the Lightweight Directory Access Protocol (LDAP). This allows Active Directory to be integrated into networks that already have a directory. It also allows you to unify and manage the multiple namespaces that may exist in the heterogeneous environments of corporate networks.

Complete Active Directory services information is available from many sources, including the Windows Server 2003 and Windows 2000 Help.

In This Section

How Host Integration Server Uses Active Directory

How to Configure Active Directory During Host Integration Server Installation

Host Integration Server Active Directory Administration

How Host Integration Server Uses Active Directory

Host Integration Server uses Active Directory by registering services and resources with the Active Directory schema. The benefits of using Active Directory include:

- Client configuration and resource location on the network is simplified.
- The limitation of 8,000 sponsor connections that existed in SNA Server 4.0 is eliminated.

Host Integration Server client computers must be configured to communicate to a Host Integration Server computer using either sponsor connections or Active Directory. A client computer cannot be set up to use both at the same time.

If a Host Integration Server computer is configured to use Active Directory, it must operate in an Organizational Unit (OU). If the Active Directory schema is new or otherwise does not have any OUs, Host Integration Server will create an OU for itself.

In addition, the following relationships between the SNA subdomain and the OU containing the Host Integration Server specific containers must be observed:

- Multiple Host Integration Server computers can exist within the same OU.
- Each OU containing a Host Integration Server must be a unique SNA subdomain.
- Two different SNA subdomains cannot exist within the same OU.
- An SNA subdomain cannot stretch across two OUs.
- If an OU contains multiple Host Integration Server computers, they must be part of the same SNA subdomain.
- The name of the OU and the name of the SNA subdomain within that OU can either be the same name or different names.

See Also

Other Resources

Active Directory Services

How to Configure Active Directory During Host Integration Server Installation

Host Integration Server participation in Active Directory is accomplished by the SNA service and the SNA Windows Management Instrumentation (SNAWMI) Provider. The SNA service is an installable option that can be selected during Host Integration Server Setup. The SNAWMI is always installed automatically when the SNA service is installed.

With Host Integration Server, there are two mechanisms that will deliver SNA client resource location. One is sponsor connections, which is the existing mechanism present in SNA Server 4.0. The second uses Windows Server 2003 or Windows 2000 Active Directory. If you want to use Active Directory instead of SNA subdomains, you must add the Host Integration Server schema to the Windows Server 2003 or Windows 2000 domain schema before installing Host Integration Server.

The Windows Server 2003 or Windows 2000 schema is extended by running a command-line utility found on the Host Integration Server CD-ROM. The following procedure adds the Host Integration Server schema to the Windows Server 2003 or Windows 2000 domain schema:

To extend the Active Directory schema

1. Make sure you are logged on to the server running Windows Server 2003 or Windows 2000 as **Administrator** or another user who is a member of both the Domain Admins and Schema Admins groups, and who has been delegated control to the domain. You can use the **Active Directory Users and Computers** administrative tool to verify user privileges and delegate control.
2. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Active Directory Users and Computers**.
3. Select **Users**, select **Domain Admins**, right-click, and then click **Properties**.
4. Select the **Members** tab and **add** members to this group.
5. Select **Users**, select **Schema Admins**, right-click, and then click **Properties**.
6. Select the **Members** tab and **add** members to this group.
7. Expand **Domain Controllers**, select the domain, right-click, and then click **Delegate Control**.
8. Complete the wizard.
9. Insert the Host Integration Server CD-ROM into the CD-ROM drive of the computer running Windows Server 2003 or Windows 2000 Server .
10. Open a Command Prompt window.
11. Change drives to the CD-ROM drive.
12. Enter the following command.

```
addschma HIserver.schema
```

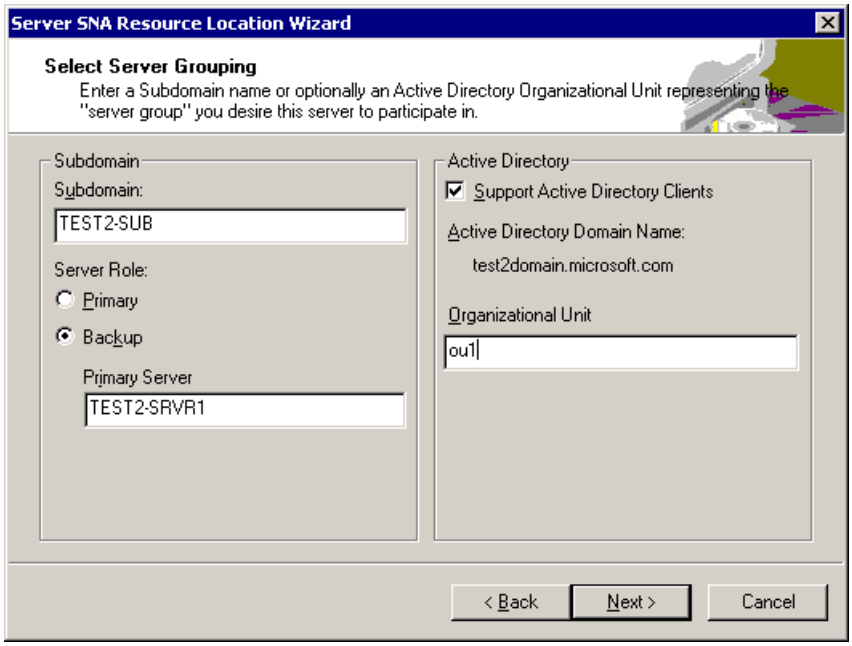
13. The resulting display will indicate if the command was successful.

Note

If you do not successfully add the Host Integration Server schema, you will not be able to use Active Directory, but you can sti ll configure a Host Integration Server client to use SNA subdomains and sponsor connections.

Configuring Host Integration Server to use Active Directory is performed during installation. The following figure shows the Setup screen where you can choose to install Active Directory support.

Select Server Grouping screen



Selecting Support Active Directory Clients, and then entering an Organizational Unit (OU) adds the following services to the ServerResources container for that OU:

- SNAServer Service
- SNAWMI Provider Service
- SNABase Service
- MngAgent Service

See Also

Other Resources

Active Directory Services

Host Integration Server Active Directory Administration

Administration of Host Integration Server Active Directory participation is done using the SNA Manager.

To access the Active Directory Server Configuration

1. In the SNA Manager tree, expand an SNA subdomain, select the server that you want to work with, right-click the server, and then click **Properties**.
2. In the **Properties** box, click **Change**. The **Server SNA Resource Location Wizard** appears.
3. Follow the on-screen prompts to install Active Directory support.

The **Server SNA Resource Location Wizard** is the same wizard that configures Host Integration Server following setup.

Note
Changes made to the **Server Properties** page require the SNABase Service to be restarted.

See Also

Other Resources

Active Directory Services

Host Configuration

For a connection to be established successfully between a host computer and a Host Integration Server computer, a number of software configuration settings (VTAM, NCP, or AS/400), and hardware characteristics must work together. These include the mainframe node ID settings, AS/400 name settings, addresses, BTU length, and link service settings.

The following table provides more details about important configuration items.

Element	Important items to consider
Host configuration settings (VTAM, NCP, or AS/400 settings) must match the **Connection and Server** settings on Host Integration Server	**Mainframe Node ID settings:** For most mainframes, IDBLK and IDNUM in the PU definition must be matched by the two parts of the Remote Node ID on the Host Integration Server connection. **AS/400 name settings:** For the AS/400, local and remote Control Point Names (CP names) and network names must match corresponding Host Integration Server settings (local names configured on the server and remote names on the connection). **Addresses:** For several connection types (802.2, X.25, and channel), host settings must be matched with equivalent settings on the Host Integration Server connection. For details, see the section about the type of connection you are configuring. **BTU length:** For the mainframe, this is set through MAXDATA in the PU definition. For the AS/400, this is set through MAXFRAME. These should equal the Max BTU Length on the Host Integration Server connection. **Other settings:** For some connections, other settings are also important. For example, for Synchronous Data Link Control (SDLC), the NRZ/NRZI settings on the host must match those on the Host Integration Server connection. For details about these and other settings, see the section about the type of connection you are configuring.
For SDLC and X.25 **Communications hardware:** line, modem (if applicable), and adapter characteristics must match the **Link service and connection** settings on the Host Integration Server	**Speed and duplexing:** For SDLC and X.25, note the speed and duplexing capabilities of the line, modem (or DCE), and adapter, and make sure that they will not be exceeded by the settings in the Host Integration Server link service and connection. Settings for fast transmission or for full duplexing cause greater demands on hardware. (Fast transmission plus full duplexing cause the greatest demands.) The hardware element with the smallest capacity will limit the capacity of the entire system. For example, with an SDLC adapter that lacks a coprocessor, you cannot use full-duplex transmissions at high speeds, even if the modem and line can handle them.

When you are configuring a new host connection or troubleshooting an existing connection, regardless of the connection type, the identifiers between the host and Host Integration Server must match. The following sections describe various configuration settings.

In This Section

Mainframe Connections Using XIDs

Node ID is the identifier used for exchange identifications (XIDs) with most mainframes. Check to make sure that the following items match; if they do not, Host Integration Server is not identifying itself in a way that the host can recognize.

Identifier used on mainframe	**Identifier to configure on Host Integration Server**
IDBLK and **IDNUM** in the PU definition	The two parts of the **Local Node ID** (configured on the connection)

See Also

Other Resources

Host Configuration

Mainframe Connections Not Using XIDs

There are some situations in which the mainframe does not use Node ID in exchange identifications (XIDs), but instead uses Network Name and Control Point Name. These situations include mainframes communicating through LU 6.2 and mainframes that call up Host Integration Server. (Host Integration Server accepts incoming calls on that mainframe connection.) In these situations, the following parameters must match.

Note
Use the following identifiers only when necessary. They are more complicated than needed for most mainframe connections and add the potential for error when used unnecessarily.

Identifier used on mainframe (in unusual cases only)	Identifier to configure on Host Integration Server
NETID in the VTAM Start command for the local SSCP	**Local Network Name** (configured on the server)
CPNAME in the PU definition	**Local Control Point Name** (configured on the server)
NETID and **SSCPNAME** in the VTAM Start command for the remote SSCP (VTAM system)	**Remote Network Name** and **Remote Control Point Name** (configured on the connection)

See Also

Other Resources

Host Configuration

AS/400 Connections

Network Name and Control Point Name (used together and called the fully qualified name) are the identifiers used when using exchange identification (XID) with AS/400 computers. Check to make sure that the following items match; if they do not, Host Host Integration Server is not identifying itself in a way that the AS/400 can recognize.

Identifier used on AS/400	**Identifier to configure on Host Integration Server**
RMTNETID (usually; often set to APPN); **RMTCPNAME**	**Local Network Name** and **Local Control Point Name** (configured on the server)
RMTNETID (often set to APPN); **CP Name** (shown in the **Display network attributes** screen)	**Remote Network Name** and **Remote Control Point Name** (configured on the connection)

Event log entries can be very helpful in diagnosing and correcting mismatched identifiers between Host Integration Server and host computers.

See Also

Other Resources

Host Configuration

Configuring VTAM for 3270 Access

When setting certain Host Integration Server parameters for a host connection, you must match values set on the host or on front-end processors for the host. Host values are configured in VTAM. Front-end processor values are configured in the Network Control Program (NCP).

The following table shows how parameters for Host Integration Server correspond to VTAM host parameters.

Host Integration Server parameter	VTAM parameter
Control Point Name (Local Node).	**CPNAME=** in the PU definition.
Network Name (Local Node) Note that the Network Name for the local and remote nodes is generally the same.	**NETID** parameter in the VTAM Start command for the local SSCP (the VTAM system to which Host Integration Server is attached). Note that this is generally the same as the NETID for the remote SSCP.

Note

The local Control Point Name and Network Name are needed only if the Host Integration Server will accept incoming calls, or will be used for LU6.2 (APPC).

See Also

Other Resources

Host Configuration

802.2 Connection Parameters

The following table shows how Host Integration Server parameters for 802.2 connections correspond to VTAM or Network Control Program (NCP) parameters. Asterisks (*) indicate required parameters.

Host Integration Server parameter	VTAM or NCP parameter
Local Node ID,* first three digits (block number).	**IDBLK=** parameter in the PU definition
Local Node ID,* last five digits (node number).	**IDNUM=** parameter in the PU definition
Network Name (Remote Node) Note that the Network Name for the local and remote nodes is generally the same.	**NETID** parameter in the VTAM Start command for the remote SSCP (VTAM system)
Control Point Name (Remote Node).	**SSCPNAME** parameter in the VTAM Start command f or the remote SSCP (VTAM system)
Remote Network Address* when connecting to a 3720, 3725, or 374 5 front-end processor.	**MACADDR=** parameter in the NCP Gen
Remote Network Address* when connecting to an IBM 9370 host.	**MACADDR**= parameter in the PORT definition
Remote SAP Address when connecting to an IBM 9370 host.	**SAPADDR**= parameter in the PU definition
Max BTU Length.	**MAXDATA**= parameter in the PU definition (set Max BTU Length ≤ MAXDATA)

* Required parameter in Host Integration Server

See Also
Other Resources
Host Configuration

SDLC Connection Parameters

The following table shows how Host Integration Server parameters for Synchronous Data Link Control (SDLC) connections correspond to VTAM or Network Control Program (NCP) parameters. Asterisks (*) indicate required parameters.

Host Integration Server parameter	VTAM or NCP parameter
Local Node ID,* first three digits (block number).	**IDBLK=** parameter in the PU definition
Local Node ID,* last five digits (node number).	**IDNUM=** parameter in the PU definition
Network Name (Remote Node) Note that the Network Name for the local and remote nodes is generally the same.	**NETID** parameter in the VTAM Start command for the remote SSCP (VTAM system)
Control Point Name (Remote Node).	**SSCPNAME** parameter in the VTAM Start command f or the remote SSCP (VTAM system)
Max BTU Length.	**MAXDATA=** parameter in the PU definition (set Max BTU Length ≤ MAXDATA)
Encoding (NRZ or NRZI).	**NRZI=** setting in the LINE/GROUP definition (defaults to YES on host)
Local Poll Address.	**ADDR**= parameter in the PU definition

* Required parameter in Host Integration Server

See Also
Other Resources
Host Configuration

X.25 Connection Parameters

The following table shows how Host Integration Server parameters for X.25 connections correspond to VTAM or Network Control Program (NCP) parameters. Asterisks (*) indicate required parameters.

Host Integration Server parameter	VTAM or NCP parameter
Local Node ID,* first three digits (block number).	**IDBLK=** parameter in the PU definition
Local Node ID,* last five digits (node number).	**IDNUM=** parameter in the PU definition
Network Name (Remote Node) Note that the Network Name for the local and remote nodes is generally the same.	**NETID** parameter in the VTAM Start command for the remote SSCP (VTAM system)
Control Point Name (Remote Node).	**SSCPNAME** parameter in the VTAM Start command f or the remote SSCP (VTAM system)
Remote X.25 Address*.	**DIALNO=** parameter in the PORT definition
Max BTU Length.	**MAXDATA=** parameter in the PU definition (set Max BTU Length ≤ MAXDATA)

* Required parameter in Host Integration Server

See Also
Other Resources
Host Configuration

Sample VTAM Parameters

VTAM parameters on host connections need to correspond to settings in Host Integration Server. Each connection needs its own specified values.

In This Section

Sample VTAM Parameters for a Token Ring Connection

The following is a sample of VTAM parameters that might be used for a Token Ring connection to an IBM 9370 host. The underlined values correspond to values specified in Host Integration Server Manager.

```
R01100   PORT
CUADDR=100,<U>MACADDR=400040004001</U>,LANCON=(5,2),X

MAXDATA=2012,MAXSTN=52,SAPADDR=04

SERVER01 PU     ADDR=C1, <U>IDBLK=05D, IDNUM=00001</U>,X

ANS=CONTINUE,<U>MAXDATA=0265</U>,MAXOUT=7,MAXPATH=7,X

PACING=0,VPACING=0,SSCPFM=USSSCS,X

LANACK=(0,0),LANCON=(5,2),LANINACT=4.8,LANRESP=(5,2), X

LANSDWDW=(7,1),LANSW=YES,MACADDR=400040001111,X

USSTAB=MSUSSTAB,DLOGMOD=D4C32792,X

PUTYPE=2,DISCNT=(NO),ISTATUS=ACTIVE,<U>SAPADDR=04</U>

T0110002 LU   <U>LOCADDR=002</U>
T0110003 LU   <U>LOCADDR=003</U>
T0110004 LU   <U>LOCADDR=004</U>
T0110005 LU   <U>LOCADDR=005</U>
P0110006 LU   <U>LOCADDR=006</U>,DLOGMOD=LU33286S
T0110007 LU   <U>LOCADDR=007</U>
T0110008 LU   <U>LOCADDR=008</U>
```

The following table shows the Host Integration Server parameters and values that correspond to the sample VTAM parameters.

Host Integration Server parameter	**Sample value**	**VTAM parameter**
Remote Network Address	400040004001	MACADDR= parameter in the PORT definition
Local Node ID, first three digits (block number)	05D	IDBLK**=** parameter in the PU definition
Local Node ID, last five digits (node number)	00001	IDNUM**=** parameter in the PU definition
Max BTU Length	265	MAXDATA= parameter in the PU definition (set Max BTU Length ≤ MAXDATA)
Remote SAP Address	04	SAPADDR= parameter in the PU definition
LU Numbers	2 through 8	LOCADDR= parameter in the LU definitions

Sample VTAM Parameters for an SDLC Connection

The following is a sample of VTAM parameters that might be used for a Synchronous Data Link Control (SDLC) connection to a host. The underlined values correspond to values specified in Host Integration Server Manager.

```
SERVER02 PU     <U>ADDR=C1,IDBLK=017,IDNUM=B8001</U>,DISCNT=NO,X
                VPACING=00,PACING=00,PUTYPE=2,<U>MAXDATA=265</U>,X
                DLOGMOD=D4C32792,USSTAB=MSUSSTAB
```

```
T01B8002 LU   <U>LOCADDR=002</U>
T01B8003 LU   <U>LOCADDR=003</U>
T01B8004 LU   <U>LOCADDR=004</U>
T01B8005 LU   <U>LOCADDR=005</U>
P01B8006 LU   <U>LOCADDR=006</U>,DLOGMOD=LU33286S
```

The following table shows the Host Integration Server parameters and values that correspond to the sample VTAM parameters.

Host Integration Server parameter	**Sample value**	**VTAM parameter**
Local Poll Address	C1	ADDR= parameter in the PU definition
Local Node ID, first three digits (block number)	017	IDBLK**=** parameter in the PU definition
Local Node ID, last five digits (node number)	B8001	IDNUM**=** parameter in the PU definition
Max BTU Length	265	MAXDATA= parameter in the PU definition (set Max BTU Length ≤ MAXDATA)
The encoding scheme (NRZ versus NRZI)	Not specified; defaults to NRZI= YES on host	The NRZI= setting in the LINE/GROUP definition
LU Numbers	2 through 6	LOCADDR= parameter in the LU definitions

Sample VTAM Parameters for an X.25 Connection

The following is a sample of VTAM parameters that might be used for an X.25 connection to a host. The underlined values correspond to values specified in Host Integration Server Manager.

```
PORTA00  PORT
CUADDR=(A00,A08),NETTYPE=1,CHARGACC=YES,CHARGE=NO,X

VCALLS=(,,001,006,,),<U>DIALNO=31370023061</U>,MAXOUT=7,
NETLEVEL=80,PMOD=8,PLENGTH=256,PWINDOW=3,
REPLYT0=3,RETRIES=7

SERVER03 PU
ADDR=C1,<U>IDBLK=05D</U>,<U>IDNUM=00025</U>,ANS=CONTINUE,X
<U>MAXDATA=265</U>,MAXOUT=7,MAXPATH=0,PACING=0,VPACING=0,X

SSCPFM=USSSCS,IRETRY=YES,USSTAB=MSUSSTAB,PUTYPE=2,X

DISCNT=YES,ISTATUS=ACTIVE
```

The following table shows the Host Integration Server parameters and values that correspond to the sample VTAM parameters.

Host Integration Server parameter	**Sample value**	**VTAM parameter**
Remote X.25 Address	31370023061	DIALNO= parameter in the PORT definition
Local Node ID, first three digits (block number)	05D	IDBLK=parameter in the PU definition

Local Node ID, last five digits (node number)	00025	IDNUM=parameter in the PU definition
Max BTU Length	265	MAXDATA= parameter in the PU definition (set Max BTU Length ≤ MAXDATA)

Sample VTAM Parameters Including CPNAME

The following is a sample of VTAM parameters that might be used for a Token Ring connection to an IBM 9370 host, when identification of the computer running Host Integration Server is based on Control Point Name (CPNAME). For this configuration, Format XIDs must be exchanged.

In the PU definition, IDBLK, IDNUM, MACADDR, and SAPADDR are not used. For this configuration to work on Host Integration Server, the Remote Node ID, Remote Network Name, and Remote Control Point Name should all be left blank. Additionally, the Local Node ID can be left at the default value (because it is not used).

In this sample, the underlined values correspond to values specified in Host Integration Server Manager.

```
R01100   PORT CUADDR=100,<U>MACADDR=400040004001</U>,LANCON=(5,2),          X
               MAXDATA=2012,MAXSTN=52,SAPADDR=04

SERVER01 PU   ADDR=C1,<U>CPNAME=SERVER01</U>,ANS=CONTINUE,                  X
              <U>MAXDATA=0265</U>,MAXOUT=7,MAXPATH=7,                       X
              PACING=0,VPACING=0,SSCPFM=USSSCS,                      X
              LANACK=(0,0),LANCON=(5,2),LANINACT=4.8,                X
              LANRESP=(5,2),LANSDWDW=(7,1),LANSW=YES,                X
              USSTAB=MSUSSTAB,DLOGMOD=D4C32792,                      X
              PUTYPE=2,DISCNT=(NO),ISTATUS=ACTIVE

T0110002 LU   <U>LOCADDR=002</U>
T0110003 LU   <U>LOCADDR=003</U>
T0110004 LU   <U>LOCADDR=004</U>
T0110005 LU   <U>LOCADDR=005</U>
P0110006 LU   <U>LOCADDR=006</U>,DLOGMOD=LU33286S
T0110007 LU   <U>LOCADDR=007</U>
T0110008 LU   <U>LOCADDR=008</U>
```

The following table shows the Host Integration Server parameters and values that correspond to the sample VTAM parameters.

Note

For this example, it is assumed that the local Network Name configured for the Host Integration Server matches the NETID parameter in the VTAM Start command for the local SSCP. (The VTAM system to which Host Integration Server is attached.)

Host Integration Server parameter	Sample value	VTAM parameter
Remote Network Address	400040004001	MACADDR= parameter in the PORT definition
Local Control Point Name (for the server)	SERVER01	CPNAME= parameter in the PU definition
Max BTU Length	265	MAXDATA= parameter in the PU definition (set Max BTU Length less than or equal to MAXDATA)
LU Numbers	2 through 8	LOCADDR= parameter in the LU definitions

See Also

Other Resources

Sample VTAM Parameters

Sample VTAM Parameters for Independent APPC

The lists in this section show sample VTAM parameters that might be used for communication using independent Advanced Program-to-Program Communications (APPC). The underlined values correspond to values specified in SNA Server Manager.

```
SERVER1  PU    ADDR=09,                                  X
               <U>IDBLK=05D</U>,                                 X
               <U>IDNUM=11111</U>,                               X
               <U>CPNAME=SNASERV</U>,                            X
               ISTATUS=ACTIVE                            X
               DLOGMOD=D4C32782,                         X
               MODETAB=ISTINCLM,                         X
               <U>MAXDATA=521</U>,                               X
               MAXPATH=1,                                X
               PUTYPE=2,                                 X
               SSCPFN=USSSCS,                            X
               MAXOUT=7,                                 X
               PASSLIM=4,                                X
               PACING=0,                                 X
               DISCNT=NO,                                X
               USSTAB=USSTAB

TRPATH    PATH DIALNO=2223331234567890,                  X
               GRPNM=AAA01,                              X
               GID=2

<U>LOCLU1</U>    LU   LOCADDR=0,                                   X
               ISTATUS=ACTIVE,<U>DLOGMOD=APPCMODE</U>,           X
               RESSCB=25,USSTAB=USSTAB,PACING=0,         X
               VPACING=0
```

In the preceding example, the LOCADDR= value for the LOCLU1 is 0. This is the correct value to use with independent APPC LUs.

The following table shows Host Integration Server parameters and values that correspond to the preceding sample VTAM parameters.

Host Integration Server parameter	**Sample va lue**	**VTAM parameter**
Connection: Local Node ID, first three digits (blo ck number)	05D	IDBLK**=** parameter in the PU definition
Connection: Local Node ID, last five digits (node number)	11111	IDNUM**=** parameter in the PU definition
Server: Control Point Name	SNASERV	CPNAME**=** parameter in the PU definition
Connection: Max BTU Length	521	MAXDATA= parameter in the PU definition (set Max BTU Lengt h ≤ MAXDATA)
Local APPC LU: LU Name	LOCLU1	Name in LU definition
APPC Partner LUs: Mode	APPCMODE	DLOGMOD= parameter in the LU definition

The following list shows mode table entries for use with independent APPC LUs.

```
*********************************************************
*           LOGMODE TABLE ENTRY FOR RESOURCES           *
*          CAPABLE OF ACTING AS LU 6.2 DEVICES          *
*********************************************************
SNASVCMG MODEENT LOGMODE=SNASVCMG,FMPROF=X'13',         X
               TSPROF=X'07',PRIPROT=X'B0',              X
               SECPROT=X'B0',COMPROT=X'D0B1',           X
```

```
                RUSIZES=X'8585',ENCR=B'0000',            X
                PSERVIC=X'06020000000000000000000300'

<U>APPCMODE</U> MODEENT LOGMODE=APPCMODE,                      X
                TYPE=0,                                  X
                FMPROF=X'13',                            X
                TSPROF=X'07',                            X
                PRIPROT=X'B0',                           X
                SECPROT=X'B0',                           X
                COMPROT=X'50B1',                         X
                PSNDPAC=X'03',                           X
                SRCVPAC=X'03',                           X
                RUSIZES=X'8585',                         X
                PSERVIC=X'06020000000000000000002F00'
```

In the preceding list, the mode named APPCMODE is an example of a mode configured for use with APPC. The mode named SNASVCMG is included in Host Integration Server, because it is required for parallel session support; it does not need to be configured in SNA Server Manager.

See Also

Other Resources

Sample VTAM Parameters

Configuring VTAM for APPC Access

Parameters on VTAM must match Advanced Program-to-Program Communications (APPC) parameters on Host Integration Server. To configure the needed parameters, consult with the host administrator for the matching VTAM parameters.

The following tables show Host Integration Server parameters and values that correspond to VTAM and CICS parameters. Note that the values given are samples only and may not work for your configuration.

The following table shows Host Integration Server parameters and values that correspond to sample VTAM parameters.

Host Integration Server parameter	**Sample value**	**VTAM parameter**
Connection: Local Node ID, first three digits (block number)	05D	IDBLK= parameter in the PU definition
Connection: Local Node ID, last five digits (node number)	11111	IDNUM= parameter in the PU definition
Server: Control Point Name	SNASERV	CPNAME= parameter in the PU definition
Connection: Max BTU Length	521	MAXDATA= parameter in the PU definition (set Max BTU Length ≤ MAXDATA)
Local APPC LU: LU Name	LOCLU1	Name in LU definition
LU 6.2 Partner LUs: Mode	APPCMODE	DLOGMOD= parameter in the LU definition

The mode named APPCMODE is an example of a mode configured for use with LU 6.2. The mode named SNASVCMG is included in Host Integration Server, because it is required for parallel session support. It does not need to be configured.

The following table shows Host Integration Server parameters and values that correspond to sample CICS parameters.

Host Integration Server parameter	**Sample value**	**CICS parameter**
LU Name for Local APPC LU	LOCLU1	Sessions
Mode name	APPCMODE	Modename
Parallel Session Limit in the mode	250	Maximum
Partner Min Contention Winner Limit in the mode	125	Maximum
Max Receive RU Size in the mode	00521	SENDSize
Max Send RU Size in the mode	00521	RECEIVESize
LU Name for remote APPC LU	CICSLU	APPLID

The parameter called Maximum has two values, 250 and 125, separated by commas. The first value (250) is the parallel session limit. The second value (125) is the host minimum contention winner limit. On Host Integration Server, this corresponds to Partner Min Contention Winner Limit in the mode. In addition, because the host is the contention winner on 125 sessions (out of a total of 250), Host Integration Server should be configured as the contention winner on the remaining 125 sessions. In this case, Host Integration Server mode would have the following values:

- Parallel Session Limit 250
- Minimum Contention Winner Limit 125
- Partner Min Contention Winner Limit 125
- Automatic Activation Limit 0

See Also

Other Resources

Sample VTAM Parameters

Sample CICS Configuration Screens for Use with APPC

The following series of screens show how CICS could be configured for independent Advanced Program-to-Program Communications (APPC) through VTAM. In the first screen, the underlined values correspond to values specified in SNA Server Manager.

```
OVERTYPE TO MODIFY                    CICS RELEASE = 0330
 CEDA  ALter
  <U>Sessions       : LOCLU1</U>
  Group          : VER3AAAA
  DEscription  ==> VERSION 3 LU 6.2 SESSION ENTRY
 SESSION IDENTIFIERS
  Connection   ==> CON1
  SESSName     ==>
  NETnameq     ==>
  <U>MOdename     ==> APPCMODE</U>
 SESSION PROPERTIES
  Protocol     ==> Appc                Appc | Lu61
  <U>MAximum      ==> 250 , 125</U>           0-999
  RECEIVEPfx   ==>
  RECEIVECount ==>                     1-999
  SENDPfx      ==>
  SENDCount    ==>                     1-999
  <U>SENDSize     ==> 00521</U>               1-30720
  <U>RECEIVESize  ==> 00521</U>               1-30720

                                           <U>APPLID=CICSLU</U>
PF 1 HELP 2 COM 3 END   6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

The following table shows Host Integration Server parameters and values that correspond to the preceding sample CICS parameters.

Host Integration Server parameter	Sample value	CICS parameter
LU Name for local APPC LU	LOCLU1	Sessions
Mode name	APPCMODE	Modename
Parallel Session Limit in the mode	250	Maximum
Partner Min Contention Winner Limit in the mode	125	Maximum
Max Receive RU Size in the mode	00521	SENDSize
Max Send RU Size in the mode	00521	RECEIVESize
LU Name for remote APPC LU	CICSLU	APPLID

In the sample CICS screen, the parameter called Maximum has two values, 250 and 125, separated by commas. The first value (250) is the parallel session limit. The second value (125) is the host minimum contention winner limit. On Host Integration Server, this corresponds to Partner Min Contention Winner Limit in the mode. In addition, because the host is the contention winner on 125 sessions (out of a total of 250), Host Integration Server should be configured as the contention winner on the remaining 125 sessions. In this case, Host Integration Server mode would have the following values:

- Parallel Session Limit 250
- Minimum Contention Winner Limit 125
- Partner Min Contention Winner Limit 125
- Automatic Activation Limit 0

The remaining screens in this section show more information about how CICS could be configured for independent APPC through VTAM.

```
OVERTYPE TO MODIFY                    CICS RELEASE = 0330
 CEDA  ALter
 SESSION PROPERTIES
  SESSPriority ==> 000                 0-255
  Transaction    :
 OPERATOR DEFAULTS
  OPERId         :
  OPERPriority   : 000                 0-255
  OPERRsl        : 0                             0-24,...
  OPERSecurity   : 1                             1-64,...
 PRESET SECURITY
  USERId       ==>
 OPERATIONAL PROPERTIES
  Autoconnect  ==> Yes                 No | Yes | All
  INservice      :                     No | Yes
  Buildchain   ==> Yes                 Yes | No
  USERArealen  ==> 000                 0-255
  IOarealen    ==> 00000 , 00000       0-32767
  RELreq       ==> No                  No | Yes
  DIscreq      ==> No                  No | Yes
  NEPclass     ==> 000                 0-255
 RECOVERY
  RECOVOption  ==> Sysdefault  Sysdefault | Clearconv
                               | Releasesess | Uncondrel
                               | None
  RECOVNotify  ==> None       None | Message | Transaction

                                          APPLID=CICSLU
PF 1 HELP 2 COM 3 END   6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

In the following screen, the underlined value corresponds to the LU Name of the local APPC LU in Host Integration Server.

```
OVERTYPE TO MODIFY                    CICS RELEASE = 0330
 CEDA  ALter
  Connection     : CON1
  Group          : VER3AAAA
  DEscription  ==> VERSION 3  LU 6.2 DEFINITION
 CONNECTION IDENTIFIERS
  <U>Netname      ==> LOCLU1</U>
  INDsys       ==>
 REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTEName   ==>
 CONNECTION PROPERTIES
  ACcessmethod ==> Vtam                Vtam | IRc
                                       |INdirect | Xm
  Protocol     ==> Appc                Appc | Lu61
  SInglesess   ==> No                  No | Yes
  DAtastream   ==> User                User | 3270 | SCs
                                       | STrfield | Lms
  RECordformat ==> U                   U | Vb
 OPERATIONAL PROPERTIES
  AUtoconnect  ==> Yes                 No | Yes | All
  INService    ==> Yes                 Yes | No
 SECURITY
  SEcurityname ==>
  ATtachsec    ==> Verify              Local | Identify
                                       | Verify | Persistent
                                       | Mixidpe
  BINDPassword ==>                     PASSWORD SPECIFIED
  BINDSecurity ==> No                  No | Yes
```

```
                                                APPLID=CICSLU
PF 1 HELP 2 COM 3 END   6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

See Also

Other Resources

Sample VTAM Parameters

Configuring NCP for Independent APPC

Parameters on Network Control Program (NCP) must match Advanced Program-to-Program Communications (APPC) parameters on Host Integration Server. To configure the needed parameters, consult with the host administrator for the matching NCP parameters.

This section provides information about NCP definitions used for supporting and defining independent LUs. The section is not intended to provide comprehensive information about NCP, which is described more thoroughly in IBM manuals such as:

- *Planning and Reference for NetView, NCP, VTAM* (CN4D1200)
- *NCP, SSP, and EP Resource Definition Guide* (CXDG7200)
- *NCP, SSP, and EP Resource Definition Reference* (CXDH1200)

You may need to study other IBM documentation as well. Some of the topics to study are:

- Independent LUs
- Type 2.1 node support
- Dynamic reconfiguration

Table of NCP Parameters That Affect Independent APPC

The following table shows basic recommendations for setting NCP parameters for independent APPC with Host Integration Server. The next sections describe the parameters in more detail.

NCP definition	NCP parameter	Recommendations
BUILD	ADDSESS	Must support the number of sessions needed.
Not applicable	AUXADDR	Must support the number of additional addresses needed.
Not applicable	MAXSESS	Use for setting the maximum sessions allowed for any LU. Corresponds to the sum of all the parallel session limits for all the modes with which a particular LU is partnered.
PUDRPOOL	NUMBER	Must support the number of servers accessing NCP.
LUDRPOOL	NUMILU	Must support the number of dynamically configurable LUs needed.
PU	NETID	Corresponds to the Local Network Name on Host Integration Server.
Not applicable	PUTYPE	For independent APPC, use PUTYPE=2.
Not applicable	XID	For independent APPC, use XID=YES.
LU	LOCADDR	For independent LUs with set definitions, use LOCADDR=0.

Not applicable	PACING	Use a value greater than or equal to the corresponding Host Integration Server parameter, the Pacing Receive Count in the mode.
Not applicable	VPACING	Use a value less than or equal to the corresponding Host Integration Server parameter, the Pacing Send Count in the mode.

BUILD Definition

The following parameters in the BUILD definition affect independent APPC:

ADDSESS= *value*

Is equivalent to the total number of sessions available to independent LUs. ADDSESS specifies the number of LU-LU session control blocks available for independent LUs. Note that session control blocks may be reserved for a particular LU by using RESSCB (reserved session control blocks) in the LU definition.

The sum of ADDSESS, AUXADDR, and NUMILU equals the total number of control blocks available for dynamic configuration of LUs. (For a description, see LUDRPOOL Definition, later in this topic.) This total should be high enough to support needed sessions, but low enough so that NCP does not exceed the storage capacity of the controller.

AUXADDR= *value*

Specifies the number of addresses that can be dynamically defined for both dependent and independent LUs. To allow for additional sessions between independent LUs, AUXADDR should be greater than ADDSESS. For more information about dynamic reconfiguration, see ADDSESS (the preceding description).

MAXSESS= *value*

Specifies the maximum number of LU-LU sessions that one independent LU can support. If you specify MAXSESS in an LU definition (as supported by NCP V6R2), the LU uses the value in the LU definition, not the BUILD definition. The limit set by MAXSESS prevents an independent LU from using too many unreserved session control blocks.

PUDRPOOL Definition

The following parameter in the PUDRPOOL definition affects independent APPC:

NUMBER= *value*

Specifies the number of physical units (PUs) that can be dynamically defined.

LUDRPOOL Definition

The following parameter in the LUDRPOOL definition affects independent APPC:

NUMILU= *value*

Specifies the number of independent LUs that can be added through dynamic reconfiguration. Find a value high enough to support needed sessions (including control-session overhead for parallel sessions), but low enough so that NCP does not exceed the storage capacity of the controller. Note that total LUs allowed by NUMILU plus NUMTYP1 plus NUMTYP2 is limited. The exact limit depends on your version of NCP.

PU Definition

The following parameters in the PU definition affect independent APPC:

NETID= *name*

Specifies the name of an adjacent network, and corresponds to the local Network Name on Host Integration Server. NETID allows the network names to differ between the host and Host Integration Server. This name is used in exchange identification (XID) negotiation.

PUTYPE=2

For independent APPC, use PUTYPE=2. When combined with XID=YES, this is equivalent to physical unit type 2.1.

XID=YES

For independent APPC, specify XID=YES, so that XIDs can be exchanged while in Normal Disconnected Mode (NDM).

LU Definition

The following parameters in the LU definition affect independent APPC:

LOCADDR=0

For independent LUs configured by an administrator (as contrasted with LUs dynamically configured by NCP), specify LOCADDR=0.

PACING= *value*

Specifies the number of frames for NCP to send to an independent LU before NCP waits for a pacing response. The value for PACING should generally be greater than or equal to the corresponding Host Integration Server mode parameter, Pacing Receive Count. This ensures a smooth flow of data from the host to the Host Integration Server. (Host Integration Server does not support adaptive pacing.)

VPACING= *value*

Specifies the number of frames for NCP to receive from an independent LU before NCP sends a pacing response; sometimes called the receive window size. The value for VPACING should generally be less than or equal to the corresponding Host Integration Server mode parameter, Pacing Send Count. This prevents delays in sending from the Host Integration Server to the host. (Host Integration Server does not support adaptive pacing.)

See Also
Other Resources
Sample VTAM Parameters

Configuring the AS/400 for 5250 Access

When setting Host Integration Server parameters for an AS/400 connection, you must match values set on the host. Check with the host administrator to obtain required information for AS/400 access.

The administrator of the AS/400 may allow configurations to be created automatically in response to incoming requests. Alternatively, the administrator may disable this feature, to ensure a higher level of security. Work with the documentation for the AS/400 or with the administrator to determine the appropriate methods for configuring the AS/400.

Note
If the administrator of the AS/400 has allowed configurations to be created automatically in response to incoming requests, t he parameters listed in this section need not be specified on the AS/400.

The following list shows command sequences on the AS/400 and the parameters to set after choosing those command sequences. You must be logged on with administrative privilege on the AS/400 to configure the listed parameters. For more detailed information, see the AS/400 documentation.

Communications > Network configuration > Configure communications and remote hardware > Work with lines

Find out the name of the line from the AS/400 administrator. To enable automatic configuration for the line, for Autocreate controller, specify *YES, and for Autodelete controller (a time-out value), specify *NONE, or a large value such as 7000 (minutes). To prevent automatic configuration for the line, for Autocreate controller, specify *NO.

If the configuration is not automatic, specify the following:

- APPN-capable *YES
- A Control Point Name and network identifier that match corresponding parameters in Host Integration Server
- APPN CP session support *YES

Communications > Network configuration > Configure communications and remote hardware > Work with communications controllers

In this context, communications controller means the Host Integration Server.

If the configuration is not automatic, specify the local address of the Host Integration Server computer.

Connecti on type	**Address as specified in AS /400 configuration**	**Method for finding out address on Host Integration Server**
802.2	LAN remote adapter address	At the command prompt, type **net config server**; then view the line labeled "Server is active on."
SDLC	Station address (STNADR)	Start Host Integration Server Manager and view the Poll Address specified for the S DLC connection. (For SDLC, both ends use same address.)
X.25	Network address	Start SNA Manager, access the **IBM X.25 Link Service Setup** dialog box, and view t he Local X.25 Address.

Guidelines for an AS/400 that Does Not Have PC Support

You can connect to an AS/400 for 5250 emulation even if the AS/400 does not have PC Support. However, to do this, you must create the mode QPCSUPP on the AS/400, along with other resources that would have been created by PC Support. A recommended way to do this is to create the QPCSUPP mode, create the QWCPCSUP class, and add the class QWCPCSUP to the system QCMN. For details about how to create these resources, see the online OS/400 documentation for the related commands: **crtmodd**, **crtcls**, and **addrtge**.

Changing Session Limits with a Single Mode Name

If the AS/400 requires different session limits on each LU (while the mode name stays the same), you must change the device description attached to the controller to reflect the correct number of sessions allowed for the LUs. For example, with a 5250

emulator, the mode name must be QPCSUPP. If you change the Parallel Session Limit on QPCSUPP for one LU-LU pair, all other LU-LU pairs that use the QPCSUPP mode will also be affected. Therefore, to change the session limit on some LUs but not others, you must change it at the AS/400.

The simplest way to get the correct number of sessions for the LUs on the AS/400 is to autocreate the controllers and devices, and then change the device description attached to the controller, so that the session limit is decreased. This is done by using the Change Device Description (**chgdevappc**) command. It needs to be done for each device description that needs to have different session limits. The AS/400 device settings that should be modified are Maximum Sessions (**maxssn**) and Maximum Conversations (**maxcnv**).

For 5250 emulation in the AS/400 environment, you must configure the following elements:

- Each Host Integration Server computer that will access the AS/400
- Connections
- Local APPC LUs
- Remote APPC LUs
- LU-LU pairs

See Also

Other Resources

Sample VTAM Parameters

Table of Parameters for AS/400 Communication

The following table summarizes details about configuring Host Integration Server for the AS/400 environment. In addition, for an X.25 connection using permanent virtual circuit (PVC), specify the PVC Alias.

SNA Server element	Parameter name in Host Integration Server	Instructions for configuring for the AS/400 environment
Server	Network Name (Local Node)	This generally corresponds to the RMTNETID setting on the AS/400. (APPN is often used.)
Server	Control Point Name (Local Node)	Corresponds to RMTCPNAME on the AS/400.
Connection	Remote End	Select Peer System as the Remote End for the connection.
Connection	Activation	Coordinate with the switched disconnect (SWTDSC) setting on the AS/400. If SWTDSC is *YES, select On Demand. If SWTDSC is *NO, select On Server Startup.
Connection	Local Node ID	Specify the Local Node ID, a required parameter for all connections. This corresponds to EXCHID on the AS/400.
Connection	Control Point Name (Remote Node)	Specify the AS/400 Control Point Name (CP name) as the remote Control Point Name for the connection.
Connection	Network Name (Remote Node)	Corresponds to the RMTNETID on the AS/400. (APPN is often used.)
Connection	802.2: Remote Network Address SDLC: Poll Address X.25: Remote X.25 Address	Corresponds to ADPTADR in the Line Description on the AS/400. Corresponds to STNADR in the Line Description on the AS/400. Specify the address of the AS/400.
Connection	Max BTU Length	Corresponds to the MAXFRAME setting on the AS/400.
Remote APPC LU	LU Name	Specify the Control Point Name of the AS/400 as the remote LU name.
Mode	Mode name	Choose the QPCSUPP mode for all LU-LU pairs to be used for AS/400 connectivity.
User or group listing	Default local APPC LU	Assign a Default local APPC LU to the user or group. If the user or group does not specify a local LU name when opening a session to an AS/400, this default local LU name will be used.
User or group listing	Default remote APPC LU	Assign a Default remote APPC LU which is the same as a default AS/400 to the user or group. If the user or group does not specify a remote LU (that is, an AS/400) when opening a session, the assigned AS/400 will be used.

See Also

Other Resources

Sample VTAM Parameters

Applications and Tools

Microsoft Host Integration Server includes several applications and tools to assist you in managing your environment. The following sections describe how to set up tracing on a Host Integration Server computer, and how to view trace files for information on the activity between components. They also discuss the performance-monitoring counters that provide basic stress testing for Host Integration Server.

In This Section

Network Management Support

The requirement to manage heterogeneous network environments containing mainframes, midrange systems, and personal computers is a big challenge. This section discusses two mainframe management tools, IBM NetView and Response Time Monitor, which are supported by Host Integration Server and can be used to manage your integrated network.

Host Integration Server can be managed either from the Host Integration Server computer (and other computers on the LAN) or from the mainframe using NetView. NetView is a centralized network management system that allows you to control SNA network resources, including Host Integration Server.

Using the IBM Response Time Monitor (RTM) provides additional management of Host Integration Server. RTM measures the length of time it takes for a host to respond to an incoming 3270 end-user request. Working in conjunction with NetView on the mainframe, RTM gathers data from 3270 terminal emulators and sends the information to the host through the NetView connection.

In This Section

How to Manage the SNA Environment Using NetView

The IBM NetView program sends and receives alerts and other information between a host (mainframe or AS/400) and the computers that connect to it. Host Integration Server supports NetView, bringing centralized SNA-based network management into local area network (LAN) environments. Using NetView, you can control any Host Integration Server computer that represents a physical unit (PU) in the SNA environment.

NetView can forward Host Integration Server alerts to the host system. In addition, NetView can be extended with the Windows NVAlert and NVRunCmd services. NVAlert provides the ability for the host system console to receive and display alerts generated by Windows and applications running on the Windows computer. NVRunCmd provides the ability to command Host Integration Server from the host system console.

The NVAlert and NVRunCmd services are installed when you install Host Integration Server. By default, the NVAlert configuration file, NVAlert.ini, is created and placed in the \Program Files\Host Integration Server\System folder. Also installed in the Windows Control Panel is the NVAlert service, and is configured with a startup setting of Manual.

If you plan to manage Host Integration Server most of the time using NetView from the mainframe, you should change the default startup configuration from Manual to Automatic. In Control Panel, click the **Services** icon, and then select the service you want to change. Click **Startup**, and then click **Automatic** as the **Startup Type**.

To configure NetView

1. In the Host Integration Server console tree, right-click the subdomain that you want to configure, and click **Properties**.
2. Click the **NetView** tab, and fill in the option.
3. Click **OK**, and save the configuration.

NVAlert runs as a background process, enabling alert messages generated by Windows or by Windows-based applications to be forwarded to a host system through NetView. To control the alert messages sent from the NVAlert service, modify the NVAlert.ini file to specify the Windows alerts that you want forwarded to the host system.

As events occur and are recorded in a log file, NVAlert compares each logged event to the list in the NVAlert.ini file. When an event matches one on the list, NVAlert forwards an alert as specified in NVAlert.ini to the host system.

If a problem occurs sending an alert, NVAlert follows a preset pattern of time-outs and retries. No additional alerts can be sent until the alert is successfully sent to the host.

See Also

Reference

NVAlert.ini File

Concepts

Sending Commands to Host Integration Server from the Mainframe

NVAlert.ini File

The following sections describe the structure and the required and optional lines of data for NVAlert.ini. For information about standard code points and values used by NetView, see your IBM NetView documentation.

Required Data Lines

You must precede each section of data lines that you add to NVAlert.ini with the following heading, including the square brackets **[ ]** as shown:

```
source.type.eventid
```

Use the Windows Event Log service to view the intended alert, and supply values as described in the following list. For information about the Event Log service, see your Windows documentation.

- For *source*, substitute the name that appears beneath the Source heading in the Windows Event Log service.
- For *type*, find the type listing (in the event detail in the Windows Event Log service), and substitute one of the following values:

Success	0
Information	1
Warning	2
Error	3
Success audit	4
Failure audit	5

- For *eventid*, use the decimal number that appears under the EventID heading in the Windows Event Log service.

Note
Choose alerts that will not exceed the basic transmission unit (BTU) of the connection over which the alerts will flow be cause the host cannot handle alerts exceeding this size.

After the heading, type data lines using the following syntax:

```
        =xx=xxxx=xxxx[,...]
```

In these lines, each letter *x* represents a hexadecimal digit specifying NetView code points. For the appropriate values to provide, see your IBM NetView documentation. The element characters [**,**...] indicate that the argument can be repeated, with commas separating the repetitions. Do not type the brackets.

Adding Optional Data Lines

To help you monitor the SNA environment or diagnose a problem, you can add additional data to the alerts sent to the mainframe. The following optional data lines can be added to an alert:

```
        =xxxx[,...]
```

```
=00,xx,string[;...]

=xxxx[,...]

=00,xx,string[;...]

=xxxx[,...]

=00,xx,string[;...]

=xxxx[,...]

=00,xx,string[;...]

=xxxx[,...]

=00,xx,string[;...]
```

Each letter *x* represents a hexadecimal digit specifying NetView code points. For the appropriate values, see your NetView documentation. The elements [**,**...] and [**;**...] indicate that the argument can be repeated, with commas or semicolons separating the repetitions. Do not type the brackets.

RecAct is an abbreviation for recommended action and *DetData* is an abbreviation for detailed data. You can use the DetData syntax to add blocks of detailed data to an alert.

- For *xx,* type the Data ID code point as described in the IBM *SNA Formats* manual (IBM document GA27-3136-12) or other NetView documentation.

- For *string*, provide a text string using the following variables:

%M	Module name.
%C	Computer name.
%I	Event ID.
%S*n*	String number *n* in the event log (*n* starts at zero).
%D*xxxx*, *yyyy*	Raw data of length *yyyy*, starting at buffer offset *xxxx*. Both *xxxx* and *yyyy* are hexadecimal values. Use this variable only if you do not specify any other variables.

Example of an Alert

The following is an example of a Windows-based alert and a description of each data line. The alert warns that a hard disk is nearly full and the description in the Windows Event Log service will read "The C: disk is at or near capacity. You may need to delete some files."

Sample alert and description

Data lines	Description
`[Srv.2.2013]`	Source is Srv, type is 2 (warning), and EventID is 2013
`AlertType=11`	Impending problem
`AlertDescription=5002`	Resource nearing capacity

`ProbableCauses=5001`	Media DASD device
`UserCauses=0102,F0A0`	Insufficient storage media space available
`UserDetData=00,95,%C`	Problem located on the indicated file server name (supplied through NetView value 95); event logged on indicated computer name (supplied through %C)

See Also

Concepts

How to Manage the SNA Environment Using NetView

Sending Commands to Host Integration Server from the Mainframe

The NVRunCmd service enables you to launch processes on the Host Integration Server from the NetView console. Commands are sent to the Host Integration Server computer, carried out, and results are returned to the NetView console as long as the results contain only alphanumeric characters. The NVRunCmd service cannot send graphical information to the mainframe NetView console. For example, if you use NVRunCmd to run the **dir** command, a character-based display of the current folder is displayed in NetView. But if you use NVRunCmd to start a graphical application, no command output is sent to NetView.

NVRunCmd does not handle back-and-forth interactions, such as a confirmation generated by a Host Integration Server computer in response to a command. If a command forwarded by NVRunCmd requires confirmation before being carried out, the command process will not be completed, and will eventually time out.

NVRunCmd is configured during Host Integration Server Setup and runs as a background process, waiting for a command to be sent from a NetView console. When NVRunCmd receives a command, it determines whether the command is one that it can process, and returns either the results (in 256-byte segments) or a messagesaying that the command failed. NVRunCmd attempts to send the results back to the NetView console only once, and if the NetView console does not receive the results, the link or some other part of the communications has failed.

The pipe symbol (|) cannot be used to combine commands. For example, NVRunCmd cannot process the following command string:

```

```

The maximum amount of information that can be returned to the NetView console is 31,700 bytes. Any additional information returned by the command will be lost, with a message appended to the truncated data warning the user.

Configuring NVRundCmd Privileges

By default, the NVRunCmd service runs in the context of the local system's security. Therefore, some commands generate responses that are different from the responses generated by a user logged on the same Windows operating system. For example, the commands **net use** and **chdir**, typed without options, generate a response in the context of NVRunCmd, not the context for a logged-on user.

For better system security, use the Windows Control Panel to access the **Services** dialog box and configure **Startup** options for the NVRunCmd service. Associate the NVRunCmd service with a user account that has the privilege level that you want for NVRunCmd.

Sending a Command from the Host Console

To send a command to the intended Host Integration Server computer, you need to specify the corresponding PU in the **runcmd** command in the NetView console of the host system. Use the following syntax to launch a process on the Host Integration Server software. Commands are not case sensitive; commands entered in lower case are converted to upper case.

```
        servicepoint


        commandname
        parameters
```

- For *servicepoint*, type the name of the VTAM PU of the Host Integration Server computer (node) in which you want the command to run. Note that on the server, this value is configured as the control point name for the server.

- For *commandname*, type the command you want to run. The command must be alphanumeric because NetView cannot return graphical output.

- For *parameters*, type the parameters needed for running the command on the server.

Note
NVRunCmd does not use or need a defined NetView connection. You can use NVRunCmd from the host to each Host Integration Server computer in the subdomain separately. The NetView connection is needed only for sending alerts to NetView.

See Also

Concepts

How to Manage the SNA Environment Using NetView

Monitoring Mainframe Response Times

The IBM response time monitor (RTM) measures the length of time it takes for a host to respond to an incoming 3270 end-user request. Responding to customer requests is a good indication of system performance. Monitoring the response time can be helpful diagnosing network and other system problems. RTM is a feature that works in conjunction with NetView on the mainframe. It gathers data from 3270 terminal emulators and sends the information to the host through the NetView connection for collection and analysis. Then you can view a graphical display of response times for a particular 3270 emulation session.

In This Section

Configuring RTM in Host Integration Server

Use the SNA manager console to configure how response times are measured and classified. Response time monitor (RTM) statistics for a specific logical unit (LU) are sent to the host that owns the LU, rather than to the connection designated for NetView data.

In This Section

How to Configure Response Time Monitor (RTM)

You can specify the times that RTM should send data and the trigger that causes it to register the host response. The configuration settings are on the **Response Time Monitor** tab of the **Properties** page for the selected SNA subdomain and are listed in the following table:

Note
For these settings to be meaningful, each emulator for your 3270 users must support RTM. If the host RTM settings are different, they override the Host Integration Server settings.

RTM Settings

Setting		**Definition**
RTM Data Sent At		Specifies when RTM data is sent to the host. You can send the data during one or both of the following situations:
	Counter Overflow	Sends the RTM data to the host when the number of host responses in a given time period overflows the size of the available counter.
	End of Session	Sends the RTM data to the host at the end of each LU-to-LU session.
RTM Timers Run Until		Specifies when the RTM registers a host response - when RTM stops the timers. (The timers are started when the local system sends data.) Possible stopping points are as follows:
	First Data Reaches Screen	Stops timing when data reaches the local screen.
	Host Unlocks Keyboard	Stops timing when the host unlocks the local keyboard.
	Host Lets User Send	Stops timing when the host lets the local computer send more data.
RTM Thresholds		Specify the cutoff times, in tenths of a second that the RTM saves its count of host responses and restarts the count. The range is 1–1000 in tenths of a second (from 0.1 seconds to 100.0 seconds). The defaults are 5, 10, 20, and 50 (0.5 seconds, 1.0 seconds, 2.0 seconds, and 5.0 seconds).

Although you can configure the RTM boundaries and definition that you want to use on your server, the host can override these values, either for an individual 3270 LU or for all LUs the host controls. The host can also specify whether to permit local display of RTM data, and, when Host Integration Server sends RTM data, it can disable collection of RTM statistics completely.

To configure Response Time Monitor (RTM)

1. In the Host Integration Server console tree, right-click the subdomain that you want to configure, and click **Properties**.
2. Click the **Response Time Monitor (RTM)** tab, and complete the options.
3. Click **OK**, and **save** the configuration.

Defining RTM Thresholds

Response time monitor (RTM) data is collected by comparing host response times against a series of four boundary values. Each time a host transaction occurs, the response time is compared with the boundary values, and the appropriate counter is incremented. There is a counter for each of the four intervals defined by the boundary values, and an overflow counter for response times above the largest boundary value.

You can change the default boundaries — 0.5, 1, 2, and 5 seconds — using the SNA manager console. You also can select which of three response time definitions to set as default.

The response time is measured from the time the user presses ENTER until one of the following events occurs:

- The first character of host data reaches the 3270 display.
- The keyboard is unlocked.
- The user is enabled to send data.

A host can override the default boundaries and other host response time settings for any or all logical units (LUs) it controls.

See Also
Concepts
Specifying When RTM Data is Sent
Indicating Lost RTM Data

Specifying When RTM Data is Sent

Response times are recorded for a specific logical unit (LU) rather than for a specific 3270 session. If a 3270 session ends and another session of the same or a different 3270 user starts using the same LU, the response time counters include responses from both sessions. To prevent the counters from collecting information from more than one 3270 session, you can reset the RTM counters on the LU before a new session begins. The counters are reset when the host requests the reset or when response time monitor (RTM) data is sent unsolicited by Host Integration Server. Usually when a host requests RTM data the counters are reset at the same time.

To ensure that RTM data relates only to the current 3270 session, you should select the **End-of-Session** checkbox when configuring RTM parameters for Host Integration Server. Note that the host can override this setting.

See Also

Concepts

Defining RTM Thresholds

Indicating Lost RTM Data

Indicating Lost RTM Data

When Host Integration Server sends response time monitor (RTM) data to the host, it can indicate to the host that RTM data may have been lost, that is, some host response times were not included in the RTM data sent to the host. This indication is sent when:

- One of the RTM counters reaches its maximum value, and Host Integration Server is not configured to send RTM data unsolicited at counter overflow. When this occurs, Host Integration Server stops recording response times until the host requests the data or until end-of-session, if unsolicited sending at end-of-session is configured. The data is sent with an indication that data may have been lost due to counter overflow.

- Connection to the host for either Host Integration Server or the Windows servers ends abnormally. The first RTM data sent to the host after the connection is re-established includes the lost-data indicator to indicate that data may have been lost due to failure of the server.

- Host Integration Server is configured to send RTM data at the end-of-session, and a new 3270 session is started before the RTM data for the previous 3270 session on the same LU can be sent. Response times for the new session are discarded until the RTM data from the previous session is sent and the counters are reset. When RTM data is sent on the new session, it includes the lost data indicator.

See Also
Concepts
Defining RTM Thresholds
Specifying When RTM Data is Sent

Additional Information About NetView and RTM

For additional information about NetView and RTM see Network Management The following topics are discussed:

- Connections used for the IBM NetView program and RTM facilities
- Link alerts
- Link statistics
- Application-generated alerts, and alerts generated through the NVAlert and NVRunCmd services
- Local logging of network management data

Using the SNA Trace Utility

The following information guides you through configuring and using the Host Integration Server SNA Trace Utility. The SNA Trace Utility is used to control trace files.

In This Section

Using the SNA Trace Utility

This section includes an overview of the Host Integration Server SNA Trace Utility and details the names and locations of the files generated by the trace application.

In This Section

SNA Trace Utility

The Host Integration Server SNA Trace Utility (snatrace.exe) is a graphical tool that allows you to enable or disable tracing options and set the SNA Trace Utility parameters.

After deciding the software components and types of tracing that can provide useful information, start the Host Integration Server SNA Trace Utility application and configure trace options for Host Integration Server computers.

If you are working to improve system performance or solve a difficulty with Host Integration Server components, reviewing trace files can assist in determining the source of the problem.

The SNA Trace Utility records activity between or within components of Host Integration Server. The files created provide detailed information about the exact sequence of events occurring within Host Integration Server components or between Host Integration Server computers and Host systems on the network.

Note
You must be an administrator on the local account to run the SNA Trace Utility.

See Also

Other Resources

Using the SNA Trace Utility

System Troubleshooting

When gathering information about system difficulties, it is best to start with information provided by the Windows Event Log and System Monitor, see Windows Utilities for additional information.

Event log information is generally more straightforward to interpret than trace information; use trace information only if event logs do not provide enough details.

When you report system problems to Microsoft Product Support (PSS), a technician may ask you for trace files and other system information:

- Trace files
- Windows Event Logs
- The current configuration file (Com.cfg)
- Dump files, which may be created by Host Integration Server when a system trap (exception error) occurs
- Datascope traces of protocol exchanges

See Also

Other Resources

Using the SNA Trace Utility

Trace and Diagnostic File Location

The following table lists the default location of files containing trace and diagnostic information:

File type	Default location	File name or file name extensions
Trace files created by the Host Integration Server Trace application	\Host Integration Server\Traces	*.atf
Log files	\WINNT\System32\Config	*.evt
Configuration file	\Host Integration Server\System\Config	com.cfg
Dr. Watson file	\WINNT	drwtsn32.log
Dump files	\Host Integration Server\Traces	snadump.log

See Also

Other Resources

Using the SNA Trace Utility

Trace File Names

Each trace file has two names associated with it, <*Filename*1>.atf and <*Filename*2>.atf.

Traces are written to the first file until it reaches the specified size, then to the second until it reaches that size, and so on alternating between the two files.

By default, the trace files are stored in the \Program Files\Host Integration Server\Traces folder, with an .atf file extension.

The following table lists the file names used by Trace:

Service	Type of tracing	File names used	File names used
SnaBase	Internal	Napint1.atf	Napint2.atf
	Message	Napmsg1.atf	Napmsg2.atf
SnaServer (PU 2.1 node)	Internal	Nodeint1.atf	Nodeint2.atf
	Message	Nodemsg1.atf	Nodemsg2.atf
Link service	Internal	Linkint1.atf	Linkint2.atf
	Message	Linkmsg1.atf	Linkmsg2.atf
NetView Alert	Internal	Nvaint1.atf	Nvaint2.atf
	Message	Nvamsg1.atf	Nvamsg2.atf
	API	Nvaapi1.atf	Nvaapi2.atf
NetView command	Internal	Nvcint1.atf	Nvcint2.atf
	Message	Nvcmsg1.atf	Nvcmsg2.atf
	API	Nvcapi1.atf	Nvcapi2.atf
SNA applications	Internal	Cliint1.atf	Cliint2.atf
	Message	Cliimsg1.atf	Cliimsg2.atf
	API	Cliiapi1.atf	Cliapi2.atf
Manage Agent	Internal	Mgaint1.atf	Mgaint2.atf
	Message	Mgamsg1.atf	Mgamsg2.atf
Manage Client	Internal	Mgcint1.atf	Mgcint2.atf
	Message	Mgcmsg1.atf	Mgcmsg2.atf
SNA Manager	Internal	Expint1.atf	Expint2.atf
	Message	Expmsg1.atf	Expmsg2.atf
	API	Expapi1.atf	Expapi2.atf
TN3270	Internal	tn3int1.atf	tn3int2.atf
	Message	tn3msg1.atf	tn3msg2.atf
	API	tn3api1.atf	tn3api2.atf
TN5250	Internal	tn5int1.atf	tn5int2.atf

	Message	tn5msg1.atf	tn5msg2.atf
	API	tn5api1.atf	tn5api2.atf
Host Print Service	Internal	sprtint1.aft	sprtint2.aft
	Message	sprtmsg1.atf	sprtmsg2.atf
	API	sprtapi1.atf	sprtapi2.atf
NetView	Internal	mnvtint1.atf	mnvtint2.atf
DDM	Internal	ddmint1.atf	ddmint2.atf
	Message	ddmmsg1.atf	ddmmsg2.atf
	API	ddmapi1.atf	ddmapi2.atf
DB2 Network Library	Internal	db2int1.atf	db2int2.atf
	Message	db2msg1.atf	db2msg2.atf
	API	db2api1.atf	db2api2.atf

See Also

Other Resources

Using the SNA Trace Utility

Choosing a Trace Type

After selecting one or more Host Integration Server components to be traced, decide the type of tracing to apply.

The following table describes the types of tracing available:

Type of tracing	Activity traced	Applies to installed components
Internal*	Activity within a software component of Host Integration Server.	SnaBase, SnaServer (PU 2.1 node), SNA applications, link services and more.
Message	Messages passed into and out of a software component of Host Integration Server, including messages sent to and received from the network.	SnaBase, SnaServer (PU 2.1 node), SNA applications, link services and more.
API	Information passed into and out of a Host Integration Server DLL, as the DLL communicates with an application, such as the APPC DLL.	SNA applications and more.

* Internal tracing is intended for use by product support technicians. Interpreting internal traces and certain types of message traces requires a specialized knowledge base.

See Also

Other Resources

Using the SNA Trace Utility

Trace Types

Before setting up tracing, decide the software components you want to trace, and which types of tracing information will be useful.

Each type of tracing is enabled using the Host Integration Server Trace application.

Internal Trace types:

- Fatal Conditions
- Error Conditions
- Debug Conditions
- Function Entry/Exit
- State Transition
- Custom Conditions

Message Trace types:

- Internal Messages
- 3270 Messages
- LU 6.2 Messages

API Trace types:

- APPC API
- CPI-C API
- LUA API
- CSV API

TN3270 Internal Trace types:

- TN3270 Internal Trace

See Also
Concepts
SNA Trace Utility

Message Traces

The following table details Message traces.

Trace option	Activity traced for Host Integration Server Applications on Host Integration Server client computers
Internal Messages	Messages between the SNA Manager (considered an application), and SnaBase and SnaServer (PU 2.1 node)
3270 Messages	Messages between 3270 applications (3270 emulators and/or LUA programs) on the local system and the PU 2.1 node
LU 6.2 Messages	Messages between the APPC DLL on the local system and the PU 2.1 node

See Also

Reference

Trace Types

Interpreting Traces

The Host Integration Server Trace feature provides message, client API, SnaBase, Transaction Integrator, PU 2.1 node, and link service tracing.

The following table shows the information needed for interpreting each type.

Type of tracing	Software component traced	Area of expertise needed
APPC API	SNA applications	APPC programming
CPI-C API	SNA applications	CPI-C programming
LUA API	SNA applications	LUA programming
SNA Formats	SnaServer (PU 2.1 Node)	SNA formats
Data Link Control (messages)	SnaServer (PU 2.1 node) or link service	DLC interface
Level 2 Messages	Link service	Link service interface
3270 Messages	SNA Applications, SnaServer (PU 2.1 Node), or SnaBase	3270 emulator interface
Internal Messages	All software components for Host Integration Server	Intended for product support personnel
APPC Messages	SNA Applications, SnaServer (PU 2.1 Node), or SnaBase	Intended for product support personnel
Internal	All software components for Host Integration Server	Intended for product support personnel

See Also

Reference

Trace Types

Using Trace to Diagnose Problems

The following table shows examples of possible difficulties and the types of tracing that may be useful.

Problem	What to trace
Host Integration Server cannot connect to host.	For the SnaServer (PU 2.1 node), trace data link control and 3270 Messages.
SNA Manager does not reflect changes in network services.	For the SnaServer (PU 2.1 node), trace Internal Messages; for Manage Agent, enable internal tracing.
Windows-based client computer cannot connect to a Host Integration Server resource.	For any client computer, enable internal tracing For a 3270 or LUA client computer, trace 3270 messages For an APPC or CPI-C client computer, trace LU 6.2 messages.

See Also

Other Resources

Using the SNA Trace Utility

Running the SNA Trace Utility

You can run the SNA Trace Utility from either the SNA Manager or from a command prompt.

In This Section

Starting the SNA Trace Utility

The following topics explain how to start the SNA Trace Utility from the SNA Manager, from the Windows **Start** menu, or from a command prompt.

In This Section

How to Start the SNA Trace Utility from the SNA Manager

The following procedure details how to start the SNA Trace Utility from the SNA Manager

To start the SNA Trace Utility from the SNA Manager

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server**, and then click **SNA Manager**.
3. Click **Tools**, and then click **SNA Trace Utility**.

How to Start the SNA Trace Utility from the Start Menu

The following procedure details how to start the SNA Trace Utility from the Windows Start menu.

To start the SNA Trace Utility from the Start menu

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server**, and then point to **Application and Tools**.
3. Click **SNA Trace Utility**.

How to Start the SNA Trace Utility from a Command Prompt

The following procedure details how to start the SNA Trace Utility from a command prompt.

To start the SNA Trace Utility from a command prompt

1. From **Start**, click **Run**.
2. Type in **command**, and click **OK**.
3. Type in:

-or-

To run the SNA Trace Utility on a specific server

1. From **Start**, click **Run**.
2. Type in **command**, and click **OK**.
3. Type in:

Tracing Servers Components

The Host Integration Server Trace application can be started locally or remotely for a Host Integration Server computer. For a client computer, the Host Integration Server Trace application can only be started locally.

Important
All procedures listed in this section assume that Trace is running on a computer that has Host Integration Server installed and configured.

In This Section

Selecting Components to Trace

Before you can begin tracing files, you need to decide on the Host Integration Server components to trace.

The Trace application enables you to record internal or external activity for the following components:

- Enterprise Single Sign-On
- NVAlert
- NVRuncmd
- SNA applications
- SNA Manager Client
- SNA Manager Agent (MngAgent)
- SnaBase
- Installed link services
- SNANetMn
- SNAPrint
- SNAServer
- SNA Management
- DB2 Network Library
- SNADDM
- TN3270
- TN5250

See Also

Reference

Tracing SNA APIs
Tracing SnaBase
Tracing PU 2.1 Node
Tracing Link Services
Tracing for TN3270
Tracing for TN5250

Tracing SNA APIs

The following table details SNA API Trace Options.

Trace option	Activity traced for SNA Applications
APPC API	Activity between APPC applications and Host Integration Server
CPI-C API	Activity between CPI-C applications and Host Integration Server
LUA API	Activity between LUA applications and Host Integration Server
CSV API	Activity between CSV applications and the CSV DLL on Host Integration Server

See Also

Reference

Tracing SnaBase
Tracing PU 2.1 Node
Tracing Link Services
Tracing for TN3270
Tracing for TN5250

Tracing SnaBase

The following table details SnaBase Trace Options.

Trace option	Activity traced for SnaBase
Internal Messages	Messages between the SnaBase software, which maintains lists of service names and statuses, the SNA Manager and Client/Server messages
3270 Messages	Messages between the SnaBase software, which maintains lists of service names and statuses, and 3270 applications (3270 emulators and/or LUA programs)
LU 6.2 Messages	Messages between the SnaBase software, which maintains lists of service names and statuses, and the APPC DLL

See Also

Reference

Tracing SNA APIs
Tracing PU 2.1 Node
Tracing Link Services
Tracing for TN3270
Tracing for TN5250

Tracing PU 2.1 Node

The following table details PU 2.1 Node traces.

Trace option	Activity traced for the SnaServer (PU 2.1 node).
Internal Messages	Messages between the SnaServer (PU 2.1 node) and the SNA Manager.
3270 Messages	Messages between the PU 2.1 node and all 3270 client computers (3270 emulators and/or LUA programs).
Data Link Control	Messages between the PU 2.1 node and link services.
SNA Formats	Data link control messages that are in Host Integration Server formats. Understanding such messages requires knowledge of Host Integration Server formats and protocols.
LU 6.2 Messages	Messages between the PU 2.1 node and the APPC DLL.

See Also

Reference

Tracing SNA APIs
Tracing SnaBase
Tracing Link Services
Tracing for TN3270
Tracing for TN5250

Tracing Link Services

The following table details link service traces.

Trace option	Activity traced for the link service
Internal Messages	Messages between the link service and the SNA Manager
Data Link Control	Activity between the link service and PU 2.1 node
Level 2 Messages	Information related to Level 2 in the international standards organization (ISO) model

See Also

Reference

Tracing SNA APIs
Tracing SnaBase
Tracing PU 2.1 Node
Tracing for TN3270
Tracing for TN5250

Tracing for TN3270

You can enable or disable tracing for TN3270 using the Host Integration Server Trace application. Host Integration Server Trace provides API, internal, and message tracing.

Host Integration Server SNA Trace Utility places trace files for TN3270 in the \Host Integration Server\Tracesfolder by default.

The following table illustrates the TN3270 file names used by Host Integration Server Trace:

Service	Type of tracing	File names used	File names used
TN3270	API	Tn3api1.atf	Tn3api2.atf
	Internal	Tn3int1.atf	Tn3int2.atf
	Message	Tn3msg1.atf	Tn3imsg2.atf
	TN3270 internal trace	Tn_00.atf – Tn0a.atf	

Note

The TN3270 internal trace contains up to 1024 bytes of information and creates up to ten files. The names are Tn_00.atf to Tn_0a.atf.

Note

SNA Trace event monitoring will not stop the TN3270 internal trace.

See Also

Reference

Tracing SNA APIs
Tracing SnaBase
Tracing PU 2.1 Node
Tracing Link Services
Tracing for TN5250

Tracing for TN5250

You can enable or disable tracing for TN5250 using the Host Integration Server Trace application. Host Integration Server Trace provides API, internal, and message tracing.

The Host Integration Server Trace application places trace files for TN5250 in the \Host Integration Server\Tracesfolder by default.

The following table illustrates the TN5250 file names used by Host Integration Server Trace:

Service	Type of tracing	File names used	File names used
TN5250	API	Tn5api1.atf	Tn5api2.atf
	Message	Tn5msG1.atf	Tn5msg2.atf
	Internal	Tn5int1.atf	Tn5int2.atf

See Also

Reference

Tracing SNA APIs
Tracing SnaBase
Tracing PU 2.1 Node
Tracing Link Services
Tracing for TN3270

Status and Performance Tools

The following information details the status and performance tools available in Host Integration Server and the Windows operating systems.

In This Section

Status and Performance Information

Host Integration Server Status

Windows Utilities

Status and Performance Information

This section details the status and performance features of Host Integration Server and the Windows operating systems.

In This Section

Status and Performance

Optimizing Performance

Network Considerations

Network Analyzer

Status and Performance

Host Integration Server and the Windows operating system offer several tools for helping you evaluate the demand on, and performance of, Host Integration Server computers and components.

You can use the following tools to quickly isolate problems:

Windows Utilities

The Windows Event Viewer allows you to monitor events on a server to troubleshoot various hardware and software problem. User Manager allows you to set up auditing and Server Manager allows you to set up administrative alerts to be sent to remote computers. For additional information on these utilities, refer to Windows documentation.

Host Integration Server Status (using the SNA Manager)

The Host Integration Server SNA Manager provides information about the current status of Host Integration Server resources, including logical units (LU), connections, and print sessions. This shared console provides a convenient and consistent environment for Host Integration Server, Transaction Integrator, and other console administration tools.

System Monitor

This Windows application enables you to measure the performance data for computers on the network. You can also monitor connections, LU sessions, and adapters.

Note
In Windows, the name of this utility is **System Monitor**, although it appears on the **Start** menu as **Performance**.

Using the SNA Trace Utility

The Host Integration Server Trace utility records activity between or within Host Integration Server components. Tracing provides detailed information of internal activities on Host Integration Server. It is helpful in isolating problems and is frequently used by product support personnel.

See Also

Other Resources

Status and Performance Information

Optimizing Performance

If more than one Host Integration Server computer is currently in use, you can evaluate the demand and performance on the servers in order to obtain a benchmark, and to estimate hardware requirements for future growth. The two primary tools for this kind of evaluation are the System Monitor, which is part of Windows, and the Host Integration Server Status available through the SNA manager.

System Monitor is a graphical tool included in Windows. The System Monitor allows you to collect performance data on your computer or on other computers in the network. For detailed instructions for using System Monitor, see the Windows documentation.

Studying activity on a moderately or heavily used server involves studying a complex set of processes and process interactions. Detailed interpretation of the data requires an understanding of how the operating system works, for example, how virtual memory is managed or how processor time is divided between competing processes.

Network Considerations

Your network adapter can have a large impact on the overall network performance. Advanced adapters provide good setup options to optimize the network I/O performance. Look for I/O buffering on the card itself, direct memory access (DMA) (adjustable data link control (DLC) (maximum frame size, and local area network (LAN) speed setting parameters.

Host Integration Server performance tuning involves adjusting the frame size, BTU size, and DLC-level pacing on the connection properties, as well as the request/response unit (RU) size and session-level pacing on the mode properties.

Additional tips:

- Your test data buffer must fit into one RU; if the RU size is too large, it will consume excess memory. 1200 bytes of screen paint data fits into an RU of 1484 bytes. This is the optimum for Ethernet. A 1920-byte data buffer would require two client/server message frames.

- The maximum BTU size needs to be at least the RU size + 9 bytes. For Ethernet, an RU of 1484 bytes and a BTU of 1493 bytes is good. For TokenRring, an RU of 4087 bytes and a BTU of 4096 bytes is a good starting point.

- The DLC level pacing is the most common tuning problem. To avoid deadlocks and timeouts, set the pacing between the two nodes so that the receiving window is one frame smaller than the partners send window. For example, node A can be set to send seven frames until it stops and waits for acknowledgement from node B; node B can be set to send an ACK after it receives six frames from node A, and vice versa. This guarantees successful DLC frame acknowledgements between the nodes.

- Setting the DLC receive acknowledgement to 1 or 2 would cause a receive ready (RR) to be sent after every second I-frame. This results in unnecessary control frame overhead on the LAN between the gateway and the server.

See Also
Other Resources
Performance Tuning

Network Analyzer

A network analyzer, is an important tool for checking the load and throughput on the network as well as verifying that there are no mismatched configurations in the test environment.

You can use a network analyzer to verify the transaction counts by monitoring the data link control (DLC) traffic to and from the computer being tested. You can filter traffic to count only incoming I-frames of a certain size and type. Using filtering, you can determine the number of accepted frames and ascertain the number of responses from the receiver servers.

Another important measurement that uses a network analyzer is following the network bandwidth usage. Ethernet, in particular, slows down drastically when usage rates grow beyond 40–50 percent. A Token Ring network is more predictable in throughput even on higher loads; however, going beyond 60 percent network load levels should be cause for attention. Finally, the network analyzer can be used to confirm proper tuning of local area network (LAN) protocols. DLC level pacing (send and receive window size), timeouts, and retransmissions could artificially limit LAN throughput, perhaps to the point that the LAN, rather than the system under test, is the performance-limiting component.

Looking at the captured messages will give a good indication of problems and where they are. If there are Receiver Not Ready (RNR) frames present, one of the nodes is overloading and then limiting the message flow in. You can minimize an excess number of Receiver Ready (RR) messages by adjusting the DLC level pacing for a larger send and receive window. Long elapsed times between the individual frames can highlight configuration problems. This indicates which of the components, client, gateway, or server, consume most of the processing time. A single client test is a good way to determine where the delays might be.

A typical time for message transmission in Host Integration Server is between 1–5 milliseconds, even on almost full CPU loads. A transmission time longer than 1–5 milliseconds indicates a configuration mismatch or lack of available RAM because the system is using the virtual memory on the hard drive.

See Also

Other Resources

Status and Performance Information

Host Integration Server Status

The Host Integration Server SNA Manager provides valuable status information for Host Integration Server computers within a SNA subdomain. You can view the status of connections, LUs, and print sessions.

In This Section

Status Information

The Host Integration Server SNA Manager provides several types of status messages that can supplement other performance information.

- You can see the status of servers, logical units (LUs), and connections (Active, Inactive, Pending, Stopping, Active [Out of Date], or Error) in the console tree by selecting the server of interest.

- You can view the number of users and the number of sessions by double-clicking the relevant server. This information can be especially useful when combined with data from System Monitor.

- Also on the SNA Manager, you can view the names of active 5250 users being supported by a particular local LU. If there is only one local LU per server, the names will include all users with active 5250 sessions on a particular server.

For example, click **Servers** in the console tree to view the status of a given SNA subdomain. The status of the servers in that subdomain will appear in the details pane.

See Also

Other Resources

Host Integration Server Status

Server Status

You can view the status of servers in a given subdomain, including the number of active users, and you can view the properties for a server to gather status information. There are six status types on the Host Integration Server SNA Manager:

- Active
- Inactive
- Pending
- Stopping
- Active [Out of Date]: Indicates that the Host Integration Server resource needs to be restarted to bring the internal parameters up to date with the latest configuration changes. First, select the affected Host Integration Server resource. Then, on the **Action** menu, click **Stop**, and then click **Start**.
- Error: Indicates that an unexpected condition has made the server inaccessible to the Host Integration Server SNA Manager.

You can also see the number of active users, 3270 sessions, Advanced Program-to-Program Communtications (APPC) sessions, and Logical Unit for Applications (LUA) sessions. If the server is active, the number of licensed users and licensed sessions is also shown.

Important
If the [Out of Date] message appears on the status bar at the bottom of the SNA Manager (as opposed to in the console tree), then the SNA Manager is out of synchronization with the rest of the SNA subdomain. The SNA Manager is out of date when another server saves the configuration file, so your copy of the file (which resides in RAM memory) is out of sync with the m aster configuration file. To synchronize your copy of the configuration file, go to the **Action** menu, and click **Refresh**.

See Also

Other Resources

Host Integration Server Status

Connection Status

The Host Integration Server SNA Manager offers six different messages that indicate the status of a connection:

- Active
- Pending
- Stopping
- Inactive
- On Demand: Indicates that the connection is available to be started when needed. On Demand connections can also have all of the statuses listed above.
- Incoming: Indicates that the connection is available to receive incoming calls. Incoming connections can also have all of the statuses listed above.

To view a connection status, simply select the appropriate connection in the SNA Manager.

See Also

Other Resources

Host Integration Server Status

Non-APPC LU Status

The status of a non-Advanced Program-to-Program Communications (APPC) logical unit (LU) can be:

- Inactive
- In Session
- System Services Control Point (SSCP). This indicates that the LU is in use, but is not yet bound to a specific host application.
- Available: Indicates the LU is recognized by the host as an available LU.
- Pending: Indicates that a user is trying to access the LU, but either the connection is inactive or the mainframe does not recognize the LU.
- Unavailable: Applies to downstream LUs only.

To view the status of an LU, select the LU in the SNA Manager.

See Also

Other Resources

Host Integration Server Status

APPC LU Status

The status of an Advanced Program-to-Program Communications (APPC) logical unit (LU) will either be Inactive or will show the number of active sessions.

To view the status of an APPC LU, select the appropriate LU in the SNA Manager.

See Also
Other Resources
Host Integration Server Status

Print Session Status

There are several different messages that show the status of a Host Integration Server print session:

- Active
- Inactive
- Pending
- In Session: Indicates the print server session is active, and the logical unit (LU) is bound. For Advanced Program-to-Program Communications (APPC), a conversation is allocated.
- Offline: Indicates the print session has been created, but the print server is not aware of it.
- Paused: Indicates the print server session is active, but the printing has been paused.

To view the status of a print session, simply select the print session in SNA Manager.

Note
The status shown by the SNA Manager and the Diagnostic tool (DISPLAY.EXE) may appear to be different for a given item (for example, LU status). This is because the SNA Manager shows the currently active sessions. The Display tool shows what was negotiated during CNOS setup.

See Also
Other Resources
Host Integration Server Status

Windows Utilities

Microsoft Windows has several built-in utilities that can assist you in tracking system usage, problems, and performance. These utilities can provide the administrator with valuable information about any computer on the network.

In This Section

Windows Event Viewer

You can use information from Windows Event Logs as you test a configuration or diagnose problems.

In This Section

Event Viewer

Windows Event Logs can tell you the sequence and type of events that led up to a particular state or situation.

The Event Logs for Windows include:

- System Event Log
- Security Event Log
- Application Event Log
- Directory Service
- File Replication Service

The Event Viewer will display the following information about system events:

- Type
- Date
- Time
- Source
- Category
- Event ID
- User
- Computer

From the View menu you can:

- View All events
- Filter Events
- View newest event first
- View oldest event first
- Find events
- Display event details
- Refresh events window.

See Also

Other Resources

Windows Event Viewer

How to Start Event Viewer

Event Logs can be viewed using the Windows Event Viewer.

You can start the Event Viewer from the SNA Manager or from the operating system.

To start Event Viewer with the SNA Manager

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server**, and click **SNA Manager**.
3. When the **SNA Manager** starts, click **Tools**.
4. Click **Event Viewer**.

To start Event Viewer with Windows

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.

See Also

Other Resources

Windows Event Viewer

How to Change Event Viewer Settings

In Windows, you can adjust Event Viewer settings by right-clicking the log and clicking **Properties**.

You can adjust the following Event Log settings:

- Maximum log size
- Overwrite events as needed
- Overwrite events older than *x* days
- Do not overwrite events (Clear log manually)
- Using a low-speed connection (Windows)

To change Event Viewer settings

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.
3. Right-click the appropriate log file (**Application**, **Security**, **System**, **Directory Service**, or **File Replication Service**).
4. Click **Properties**.

See Also
Other Resources
Windows Event Viewer

How to Save Event Logs

You can save event logs for later reference or for historical data. Event log files can be saved as event files (*.evt), text files (*.txt). or comma-delimited text files (*.txt).

To save event logs

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.
3. Right-click the appropriate log file (**Application**, **Security**, **System**, **Directory Service**, or **File Replication Service**).
4. Click **Save Log File As**.
5. Type a name for the file, and click **Save**.

See Also

Other Resources

Windows Event Viewer

How to Clear Event Logs

In Windows, you can clear the event logs by selecting **Clear all Events** on the **Action** menu after selecting the appropriate log file. You have the option of saving the event log before you clear it.

To clear event logs

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.
3. Select the appropriate log file (**Application**, **Security**, **System**, **Directory Service**, or **File Replication Service**).
4. Click the **Action** menu, and then click **Clear all Events**.
5. You will be prompted to save the file.

Note
There is no verification for clearing the event log.

See Also

Other Resources

Windows Event Viewer

How to Select Computers in Event Viewer

In Windows, you can select any computer in your network to view its event logs in Event Viewer.

To select computers in Event Viewer

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.
3. Right-click **Event Viewer** (top level).
4. Select **Connect to another computer**.
5. Type the computer name on which to view Event Logs, and click **OK**.

See Also

Other Resources

Windows Event Viewer

How to Filter Events

In Windows, you can specify the type of information you want the event logs to record. The information can include the following:

- Event types
- Event source
- Category
- Event ID
- User
- Computer
- Events for various time/dates

To filter events

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Event Viewer**.
3. Right-click the appropriate log file (**Application**, **Security**, **System**, **Directory Service**, or **File Replication Service**).
4. Select **Properties**, and click the **Filter Tab**.
5. Type the appropriate information that you want to filter, and then click **OK**.

See Also

Other Resources

Windows Event Viewer

How to Find Events

You can specify what type of event to find based on the following criteria:

- Event type
- Source
- Category
- Event ID
- Computer
- User
- Description
- Direction (up or down from the currently selected event)

How to find events

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Event Viewer**.
2. Right-click the appropriate log file (**Application**, **Security**, **System**, **Directory Service**, or **File Replication Service**).
3. Click the **View** menu, and select **Find**.
4. Enter the appropriate information that you want to find.
5. Click **Find Next**.
6. Click **Close** when you are finished.

See Also

Other Resources

Windows Event Viewer

Setting Audit Policy

You can set the audit policies within the operating system. The audit policies control which events will be logged to the event log files.

In This Section

Audit Policies

How to Set Audit Policy

Audit Policies

Auditing security events and then placing entries in the computers security log can track selected activities of users. Use the audit policy to determine the types of security events that are logged.

Because the security log is limited in size, choose to log only those events necessary. The maximum size of the computer's security log is defined in Event Viewer.

Entries in a security log can be reviewed using Windows Event Viewer.

How to Set Audit Policy

The following procedure details how to set an audit policies with Windows.

To set an audit policy

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Local Security Policy**.
3. Expand **Local Policies**, and select **Audit Policy**.
4. Make appropriate changes to audit policies.

See Also

Concepts

Audit Policies

System Monitor

This section details the Windows System Monitor.

In This Section

System Monitor Overview

Using the Windows System Monitor, you can view reports on CPU load, memory usage, and interrupt rate, as well as the overall throughput of Host Integration Server traffic on the network. Restrained use of the system monitor is recommended because the tool itself can cause extra stress on the servers CPU. Specifically, this can happen if tracking all the details of many logical units (LUs) over the network from another server. Try to limit the system monitor to providing summary statistics only.

Also, try to check the CPU loads of your receiver servers and stress client computers during a practice run to make sure they are not overloaded. If the client computers are Windows computers, you can use the System Monitor application in Windows to check the CPU load on the client machines.

Host Integration Server services are fully integrated with the Windows operating system. This allows the services, connections, and processes associated with Host Integration Server to be assigned to the System Monitor. You can evaluate the demand and performance of one or more Host Integration Server-based computers to obtain a benchmark and to estimate hardware requirements for future growth.

You can use the Windows System Monitor to look at the resource use of specific components and program processes. With the system monitor, you can create charts and reports that gauge a computer's efficiency; identify and troubleshoot possible problems such as unbalanced resource use, insufficient hardware, or poor program design; and plan for additional hardware needs.

Using the System Monitor, you can configure object counters and instances to assist in evaluating Host Integration Server performance. Specific counters and instances appear when a particular service is installed and running on the server.

For detailed instructions about using the System Monitor, see your Windows Server documentation.

See Also

Tasks

How to Start System Monitor
How to Configure System Monitor
How to Save Performance Data

Concepts

Useful Performance Counters
Performance Counters on Transaction Integrator
Maximizing Communications Throughput

Useful Performance Counters

Performance counters allow you to see where computer resources are being used. The counters described below provide valuable information for evaluating the demand on, and performance of, Host Integration Server components.

- **Memory: Pages/sec**

  In order to understand memory load on a Windows Server, you must first understand paging, which is a technique for implementing virtual memory. Paging is switching blocks (pages) of program instructions or data back and forth between memory and disk. Paging is carried out as needed by the virtual memory manager in Windows.

  Pages/sec is the number of pages read from the disk or written to the disk because they were not in memory when needed (that is, the number of page faults that required disk access). The counter includes paging traffic generated when the cache accesses file data for applications.

  Pages/sec is the primary counter for determining whether your server is paging excessively. As this counter goes up, server responsiveness slows because of the time required for disk access (reading or writing). A server dedicated to communications should be equipped with enough physical memory so that little paging is required.

  The highest acceptable value for pages/sec varies from system to system. One way to judge whether system load is causing too much paging is to observe whether processor activity drops significantly as paging increases. This indicates that the system is occupied with switching pages rather than with actually processing instructions.

  The primary ways to correct excessive paging are to add more physical memory to the server or to decrease the demand on the server. Demand can be decreased by narrowing the variety of tasks that a server must perform or by decreasing the number of users accessing a server. For example, an overloaded multipurpose server with file, print, and Host Integration Server demands, could be dedicated to Host Integration Server traffic only, or user loads placed on one server could be divided between two servers (load balancing). In any case of memory overload, adding physical memory may provide the needed performance increase.

  A secondary way to decrease the impact of paging is to upgrade the disk system. This includes installing a faster disk, installing a second disk, using RAID striping, or similar upgrades. This upgrading does not decrease paging (pages/sec), but speeds up the paging process itself. For example, replacing a slow IDE disk with a faster SCSI disk may make a given paging rate, perhaps 20 – 40 pages/sec acceptable.

- **System: %Total Processor Time and Processor: %Processor Time**

  System: %Total Processor Time is the percentage of elapsed time during which the system processors are busy. It can be viewed as the fraction of total processor time spent doing useful work. Values of 60 – 80 percent during typical loads are good values because they allow some reserve for peak loads. However, when the processor stays at 100 percent for periods of time, this may indicate a processor bottleneck. On a multiprocessor system, you can view **Processor: %Processor Time** for each processor to see how the load is distributed among processors.

  One useful way to view **Total Processor Time** values is in Chart view, along with counters indicating increases and decreases in user load. For user load, such counters include **Host Integration Server Logical Unit Sessions: Throughput Bytes/Sec**, and **Host Integration Server Adapter <*adaptername*>: Throughput Frames/Sec.** These two counters are available only when there is Host Integration Server activity. For example, you might notice that during a period of peak demand for logical unit sessions, Total Processor Time reaches 100 percent and stays there. This could indicate that the Host Integration Server computer is reaching peak capacity and that any additional demand might require additional processors or additional servers.

  It may also be helpful to view **System: %Total Processor** Time along with any other counters related to the servers major functions. For example, when a Host Integration Server computer is also a file server, the **Server: Server Sessions** counter can be helpful. Other counters that may help you analyze the sources of processor activity are **Process: %Processor Time** for processes you think are relevant, as well as **System: Total Interrupts/sec**. Also, if your client computers are using NetWare, consider looking at **NWLink SPX: Connections Open**. Similarly, if your client computers use TCP/IP, consider looking at **TCP: Connections Established**.

- **System: Total Interrupts/sec and Processor: Interrupts/sec**

  System: Total Interrupts/sec is the rate at which the computer is receiving and servicing device interrupts. Device interrupts are the signals that a device sends to a processor to indicate that a task is complete or the device requires attention. Some devices that may generate interrupts are adapters, network adapters, the system timer (clock), and the mouse. **System: Total Interrupts/sec** provides an indication of how busy these devices are on a computer-wide basis.

  Similarly, for each processor, **Processor: Interrupts/sec** is the rate at which the processor is receiving device interrupts.

  Normal thread execution is suspended during interrupts. An interrupt may cause the processor to switch to another, higher-priority thread. Clock interrupts are periodic and frequent (on the order of 100 per second); they create a background of interrupt activity.

  These counters can help indicate the general demand on a server, and may be useful when combined with processor and memory data, such as **System: %Total Processor Time and Memory: Pages/sec.**

- **SNA Connections: Throughput Bytes/Sec**

- **SNA Logical Unit Sessions: Throughput Bytes/Sec**

- **SNA Adapter adaptername: Throughput Frames/Sec**

  These counters provide an indication of Host Integration Server activity. When observing these counters, it may also be useful to start the SNA Manager and double-click the same server being observed in System Monitor. You can see the number of users and sessions that correlate with a particular level of Host Integration Server activity. This information, combined with data about the processor and memory load, can help you understand the load and performance on your servers. Low throughput does not necessarily mean poor performance, but instead may simply indicate that current activity is low.

  Measurement of frames/second may provide a better indicator of server load than bytes per second, because server overhead for interrupt handling and message processing increases per frame, not per byte. In other words, a large frame with many bytes requires about the same overhead as a small frame with fewer bytes.

- **SNA Connections: Data Bytes Received/Sec**

- **SNA Connections: Data Bytes Transmitted/Sec**

- **SNA Logical Unit Sessions: Data Bytes Received/Sec**

- **SNA Logical Unit Sessions: Data Bytes Transmitted/Sec**

- **SNA Adapter adaptername: Data Bytes Received/Sec**

- **SNA Adapter adaptername: Data Bytes Transmitted/Sec**

- **SNA Adapter adaptername: Frames Received/Sec**

- **SNA Adapter adaptername: Frames Transmitted/Sec**

- **SNA Adapter adaptername: Throughput Bytes/Sec**

  These counters provide additional detail about Host Integration Server activity when used with the previous three counters.

- **SNA Adapter adaptername: Adapter Failures**

- **SNA Adapter adaptername: Connection Failures**

- **SNA Adapter adaptername: Successful Connects**

  These counters may be useful for detecting patterns in which connections or adapters fail for short periods and then return to normal. Event Logs can provide more information about causes of failure. You might also want to set up System Monitor alerts with these counters, so that an alert is triggered if too many failures occur.

See Also

Concepts

System Monitor Overview

Performance Counters on Transaction Integrator

The following performance counters are available for Transaction Integrator.

Average Method Call Time
This counter measures the average time it takes Transaction Integrator to process method calls made by the client application. The time begins when Transaction Integrator recognizes the request from the client application (the Invoke call). The time ends when Transaction Integrator returns control to the client application. This counter is not specific to any Transaction Integrator programming model. This counter is represented in terms of seconds of elapsed time.

Bytes received from host/sec
This counter indicates the number of bytes received from the mainframe by Transaction Integrator. This counter is not specific to any Transaction Integrator programming model. For the CICS Link model, the number reported will be slightly more than the amount of user data due to link model protocol header data. This number is represented in terms of bytes per second.

Bytes sent to host/sec
This counter indicates the number of bytes sent from Transaction Integrator to the mainframe. This counter is not specific to any Transaction Integrator programming model. For the CICS Link model, the number reported will be slightly more than the amount of user data due to link model protocol header data. This number is represented in terms of bytes per second.

Host response time CICS Link
This counter measures the average time the host spends processing the transaction programs unit of work when the CICS Link model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after Transaction Integrator sends the final data buffer and ends when the first response buffer is received by Transaction Integrator. This counter is represented in terms of seconds of elapsed time.

Host response time CICS Non-link or IMS
This counter measures the average time the host spends processing the transaction programs unit of work when either the CICS Non-link or IMS models are being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after Transaction Integrator sends the final data buffer and ends when the first response buffer is received by Transaction Integrator. This counter is represented in terms of seconds of elapsed time.

Link calls/sec
This counter measures the number of method calls that use the CICS Link programming model. This number is in terms of calls per second.

Non-link calls/sec
This counter measures the number of method calls that use the CICS Non-link or IMS programming model. This number is represented in terms of calls per second.

Total calls/sec
This counter indicates the total number of method calls that Transaction Integrator has processed. This counter is not specific to any Transaction Integrator programming model. This number is represented in terms of calls per second.

Total errors/sec
This counter indicates the total number of method calls that have returned a non-zero HRESULT indication to the client application. This counter is not specific to any Transaction Integrator programming model. This number is represented in terms of errors per second.

See Also
Other Resources
Transaction Integrator Performance Guide

Maximizing Communications Throughput

Servers used primarily for communications need to provide fast throughput, but do not need to provide fast file access as a file server would. Faster throughput will result if portions of memory are set aside for communications programs such as Host Integration Server or Microsoft SQL Server.

Such dedicated memory includes nonpaged memory, or portions of memory that are never switched to disk, but remain available for immediate use at all times. This helps support fast throughput. If more memory is dedicated to Host Integration Server or similar programs, less memory is available for file sharing.

With Windows operating systems, you can view or change network throughput options. For Host Integration Server, you may not need to change the throughput option. Setup automatically sets the option to maximize throughput for network applications.

Servers used primarily for communications run many important background processes. Background processes are processes not related to user actions in the current window. These servers generally do not need to run foreground processes at maximum speed. Therefore, making the operating system more responsive to background processes and somewhat less responsive to foreground processes can increase Host Integration Server throughput. Setting background or foreground responsiveness is known as tasking.

A server that is less responsive to foreground processes will run local applications such as word processing software, spreadsheets, or the SNA Manager more slowly. Tasking is most appropriate for servers used primarily to support client computers, not servers used locally as desktop computers.

The options available for tasking are:

- Best Foreground Application Response Time
- Foreground Application More Responsive than Background
- Foreground and Background Applications Equally Responsive

Choose the one that is best suited to your network configuration.

See Also
Concepts
System Monitor Overview

How to Start System Monitor

You can start the Windows System Monitor in several ways. The following procedures describe how to start the system monitor from the operating system and from the SNA Manager.

To start System Monitor with the SNA Manager

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server**, and click **SNA Manager**.
3. Once the **SNA Manager** starts, click **Tools**.
4. Click **System Monitor**.

To start System Monitor with Windows

1. Click **Start**, and point to **Programs**.
2. Point to **Administrative Tools**, and then click **Performance**.

See Also

Concepts

System Monitor Overview

How to Configure System Monitor

Configuring System Monitor consists of adding counters to the System Monitor user interface. You can also set up administrative alerts to be generated by System Monitor. For more information about using System Monitor, see the Windows operating system Help.

The following procedure details how to add counters to System Monitor with the Windows operating system.

To add System Monitor counters

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server**, and click **SNA Manager**.
3. Once the **SNA Manager** starts, click **Tools**.
4. Click **Performance Monitor**.
5. Click **Add** (Plus sign).
6. Select the object for which to gather performance data (**SNA Connections** for example).
7. Select the counter.
8. Select the instance.
9. Click **Add**.
10. Repeat steps 6 – 9 for each counter you add.
11. Click **Done** to return to the **Performance Monitor**.

Note
Using Windows, if you load System Monitor from a Terminal Server client session, the performance counters for Host Integration Server and the MSMQ-MQSeries Bridge do not appear. When you are using the System Monitor in Terminal Server client session, you have to use "\\ComputerName\" instead of "\\ComputerName" for the computer field in order to be able to access the Host Integration Server or MSMQ-MQSeries bridge counters.

See Also

Concepts

System Monitor Overview

How to Save Performance Data

When you capture performance data, you can save the data for future use.

To save System Monitor data

1. Click **Start**, and point to **Programs**.
2. Point to **Host Integration Server** and then click **SNA Manager**.
3. Once the **SNA Manager** starts, click **Tools**.
4. Click **Performance Monitor**.
5. Click **Add** (Plus sign).
6. Select the object for which to gather performance data (**SNA Connections** for example).
7. Select the counter.
8. Select the instance.
9. Click **Add**.
10. Repeat steps 6 through 9 for each counter you add.
11. Click **Done** to return to the **System Monitor**.
12. Collect the desired data, click **Console**, and then click **Save As**.
13. Type a file name, and click **Save**.

See Also
Concepts
System Monitor Overview

Performance Tuning

In Windows, there are only two parameters that have to be set for optimum performance: the *application performance boost* and the *system performance balance*.

In This Section

How to Boost Application Performance with Windows

The following procedure details how to set the parameters for the application performance boost.

To boost application performance with Windows

1. In the Windows Control Panel, double-click **System**.
2. On the **Advanced** tab, click **Performance Options**.
3. Select the **Application response** (Application or Background services) and click **OK**.

See Also

Tasks

How to Balance System Performance with Windows

How to Balance System Performance with Windows

The following procedure details how to set the parameters for the system performance balance.

To balance system performance with Windows

1. In the Windows Control Panel, double-click the **Network and Dial-up Connections** icon, and then double-click **Local Area Connection**.
2. Click **Properties**.
3. Select **File and Printer Sharing for Microsoft Networks**, and double-click **Properties**.
4. Select the optimization setting, and click **OK**.

See Also

Tasks

How to Boost Application Performance with Windows

Messaging User's Guide

Microsoft MSMQ-MQSeries Bridge is an adaptable system that can be customized. You can set up MSMQ-MQSeries Bridge to operate on almost any Message Queuing (also known as MSMQ) or IBM MQSeries network configuration.

In This Section

Using MSMQ-MQSeries Bridge

The topics in this section describe the procedures for using MSMQ-MQSeries Bridge. In addition, Help buttons located on the MSMQ-MQSeries Bridge user interface are linked to topics specific to their context.

In This Section

MSMQ-MQSeries Bridge Overview

MSMQ-MQSeries Bridge is an external gateway between two otherwise incompatible message queuing systems. MSMQ-MQSeries Bridge provides a seamless interface between Message Queuing on computers running Microsoft Windows Server 2003 or Windows 2000 Server and IBM MQSeries running on mainframes and other systems.

See Also

Other Resources

Using MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Operation

MSMQ-MQSeries Bridge is an interface between Message Queuing and MQSeries, so that you can send messages between Message Queuing applications and MQSeries queues.

MSMQ-MQSeries Bridge operates entirely in the background. In the MSMQ to MQSeries direction, a Message Queuing application can send a message to an MQSeries queue by a standard MQSendMessage() application programming interface (API) call or ActiveX control.

An MQSeries application can receive the message from the MQSeries queue by a standard MQGET() API call. In the MQSeries to MSMQ direction, the opposite relations apply. Neither application needs to be aware that the message has crossed between environments.

The MSMQ-MQSeries Bridge interface extends the features of Message Queuing and MQSeries across the combined environment. MSMQ-MQSeries Bridge fully supports connectionless, asynchronous messaging.

Using MSMQ-MQSeries Bridge, you can route messages to each messaging system even if the two systems are not connected to the network at the same time.

See Also

Other Resources

Using MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Benefits

Using MSMQ-MQSeries Bridge, your applications can send messages between IBM MQSeries and Message Queuing easily and efficiently. MSMQ-MQSeries Bridge extends connectionless, store-and-forward messaging across messaging systems and computing platforms throughout your network.

MSMQ-MQSeries Bridge offers:

Compatibility
Each messaging system sends and receives data in its native format. MSMQ-MQSeries Bridge converts the message formats automatically between the systems.

Adaptability
Your applications can send messages using standard Message Queuing or MQSeries API calls. You do not need to recode existing applications to use MSMQ-MQSeries Bridge.

Reliability
MSMQ-MQSeries Bridge supports transactions and deliver-once features, ensuring that messages are properly delivered following recovery from a system failure.

Performance
You can customize MSMQ-MQSeries Bridge for optimal performance in your network environment.

Management
Using MSMQ-MQSeries Bridge Manager, you can configure and manage all MSMQ-MQSeries Bridge computers in your enterprise network from a central location.

See Also
Other Resources
Using MSMQ-MQSeries Bridge

Message Queuing and MQSeries Features

The Message Queuing and MQSeries message queuing systems offer the following features:

Connectionless, asynchronous messaging
The communicating applications do not need to log on to the remote system or establish a session with each other. The computers on which the applications run do not need to be connected at the instant when messages are written or read. Applications can continue running without waiting for transmission to be completed.

Guaranteed delivery and deliver once
Message Queuing and MQSeries provide mechanisms by which an application can guarantee and confirm that messages are delivered, and prevent duplicate delivery.

Message prioritization
A sending application can specify the order in which the receiving application will get the messages.

User-defined message structure
The message body may contain a single byte (or no message contents at all), a text string, or a long and complex data structure. The message body may be structured or encrypted in any syntax that the communicating applications understand.

Transaction support
Send-message or receive-message operations can participate in a transaction, and can be coordinated with other operations such as database updates. The entire transaction is canceled and rolled back if any of the operations fail.

Application programming interface (API)
MSMQ and MQSeries operate on the Application Layer of the ISO Reference Model for Open System Interconnection. They act as a simple interface between an application program and the network, freeing the application programmer from concern about network or communication details.

See Also
Other Resources
Using MSMQ-MQSeries Bridge

Reference Material

Before installing or programming MSMQ-MQSeries Bridge, you should be familiar with the principles of message queuing and with at least one message queuing environment (Message Queuing or MQSeries).

For background information, refer to the *Host Integration Server Programmer's Guide*, Message Queuing documentation, and the IBM MQSeries documentation.

See Also

Other Resources

Using MSMQ-MQSeries Bridge

How MSMQ-MQSeries Bridge Works

The following topics detail how MSMQ-MQSeries Bridge works.

In This Section

MSMQ-MQSeries Bridge Concepts

With MSMQ-MQSeries Bridge, Message Queuing (also known as MSMQ) and MQSeries applications can send messages to each other, between the message queuing systems.

MSMQ-MQSeries Bridge achieves this by mapping the messages and the data fields of the sending system and the values associated with those fields, to the fields and values of the receiving environment.

After the mapping and conversion, MSMQ-MQSeries Bridge completes the process by routing the message across the combined Message Queuing and MQSeries networks.

See Also

Other Resources

How MSMQ-MQSeries Bridge Works

Message Queuing Concepts

Message queuing enables programs to share data across a network without necessarily having a synchronized connection linking the sending and receiving applications at the same instant. The programs do this by putting the data, or message, on a message queue, which is then retrieved by the receiving application.

Two basic concepts of both Message Queuing and MQSeries are message and message queue:

Message
A message is a set of data that needs to be transmitted from one application to another application, on the same or a different computer in a network.

Message queue
A message queue is the location where messages are stored, which can be written and read by applications.

See Also
Other Resources
How MSMQ-MQSeries Bridge Works

Message Fields or Properties

A message may contain one or more fields, such as the message buffer or body, label, priority, or sender ID. The following applies to fields or properties:

- In MQSeries, the fields are members of a fixed data structure.
- In Message Queuing, the fields are known as properties. A message can contain any number of properties (even zero). In practice, an application assembles a message from one or more properties in a dynamic data structure.

See Also

Other Resources

How MSMQ-MQSeries Bridge Works

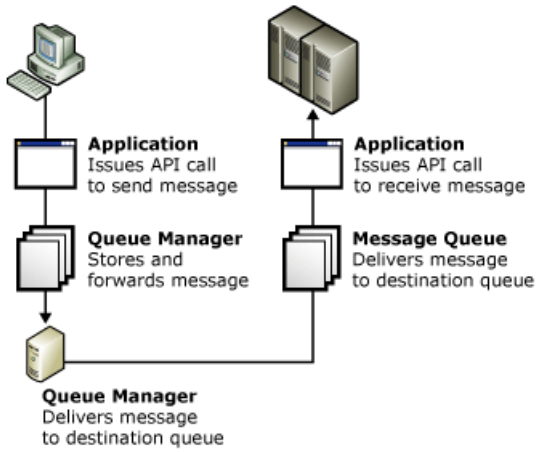
Sending and Receiving Messages

To create a message, an application specifies the message fields or properties and supplies the field values. The application then issues a Message Queuing or MQSeries API call to send the message.

The Message Queuing or MQSeries Queue Manager (server) transmits the message to the destination message queue. If the destination location is not connected to the network when the message is sent, the message queuing system stores the message at an interim location. The system forwards the message automatically when a connection is established.

To receive a message, an application issues an API call that reads the message from the queue.

Receiving a message



See Also

Other Resources

How MSMQ-MQSeries Bridge Works

System Components

The MSMQ-MQSeries Bridge system can contain two main components:

MSMQ-MQSeries Bridge
Converts and transmits messages between the Message Queuing and MQSeries environments.

MSMQ-MQSeries Bridge Manager
Configures, monitors, and controls the messaging traffic through MSMQ-MQSeries Bridge.

See Also
Other Resources
How MSMQ-MQSeries Bridge Works

Message Conversion

MSMQ-MQSeries Bridge maps the fields or properties of a message to the corresponding fields or properties of the destination message queuing system. For example, if you send a message from MQSeries to Message Queuing, MSMQ-MQSeries Bridge analyzes the fields of the MQSeries message and maps each value to its Message Queuing counterpart. In cases where one system needs an additional field that does not exist in the other, MSMQ-MQSeries Bridge provides the field during the conversion process.

For example, if a Message Queuing message includes the PROPID_M_TIME_TO_BE_RECEIVED property with a specific value, MSMQ-MQSeries Bridge maps this property to the MQSeries MQMD.Expiry property and multiplies the value by 10 to change the units from seconds to tenths of seconds.

MSMQ-MQSeries Bridge does not restrict the content of a message. The message body may contain its own internal structure, which is recognized only by the sending and receiving applications and is not interpreted in any way by MSMQ-MQSeries Bridge.

For detailed information about how MSMQ-MQSeries Bridge maps and converts properties from Message Queuing to MQSeries and from MQSeries to Message Queuing, see the *Host Integration Server Programmer's Guide*.

See Also

Other Resources

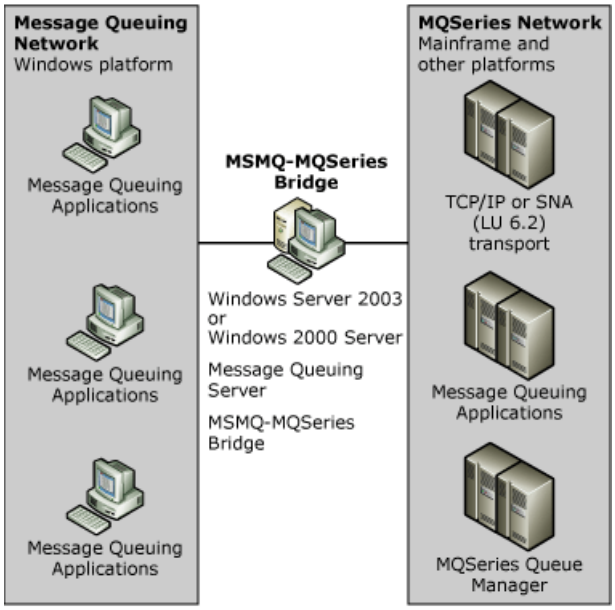
How MSMQ-MQSeries Bridge Works

Network Architecture

MSMQ-MQSeries Bridge is used in conjunction with Message Queuing and MQSeries networks. MSMQ-MQSeries Bridge is a Message Queuing Connector application.

MSMQ-MQSeries Bridge is installed on a Microsoft Windows Server 2003, Enterprise Edition or Windows 2000 Server Enterprise Edition system that serves as a connection point between the networks. A Message Queuing routing server must be installed on the same computer as MSMQ-MQSeries Bridge, and the computer must be connected by a TCP/IP or LU 6.2 link to an MQSeries Queue Manager.

MSMQ-MQSeries Bridge connects Message Queuing and MQSeries



See Also

Other Resources

How MSMQ-MQSeries Bridge Works

Multiple Connections

You can connect any number of Message Queuing or MQSeries systems or networks using MSMQ-MQSeries Bridge. For example, you can connect:

- A single MSMQ-MQSeries Bridge to several MQSeries Queue Managers
- A single MQSeries Queue Manager to several MSMQ-MQSeries Bridges
- Several MSMQ-MQSeries Bridges to several MQSeries Queue Managers

See Also
Other Resources
How MSMQ-MQSeries Bridge Works

Sending Messages From Message Queuing to MQSeries

To send a message from Message Queuing to MQSeries, you must define a Message Queuing foreign computer representing the MQSeries Queue Manager, and the MQSeries destination queue must already exist. If this is not the case, see Installing and Configuring MSMQ-MQSeries Bridge.

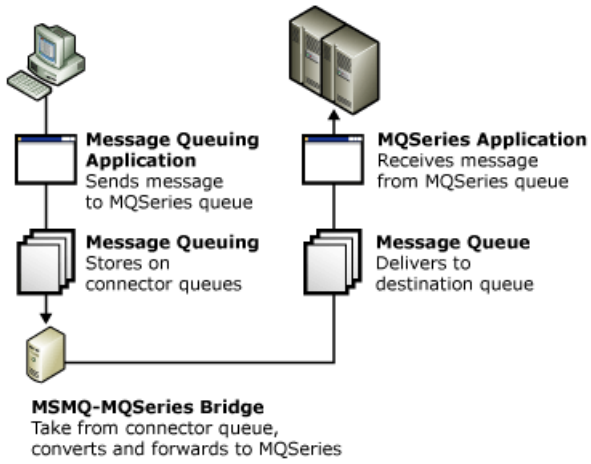
The messaging process is as follows:

- A Message Queuing application issues a Message Queuing MQCreateQueue() API call to create a foreign queue, located on the foreign computer and representing the MQSeries destination queue. Alternatively, you can create the foreign queue using Message Queuing, and in Windows Server 2003 or Windows 2000 the foreign queue is a part of Users and Computers.

- The application calls MQOpenQueue() to open the foreign queue.

- The application calls MQSendMessage() to send a message to the foreign queue. Message Queuing routes the message and stores it temporarily on a Message Queuing connector queue.

  Sending a message from Message Queuing to MQSeries



- MSMQ-MQSeries Bridge takes the message from the connector queue and converts the message properties to the MQSeries message structure. MSMQ-MQSeries Bridge routes the message to the MQSeries destination queue.

- An MQSeries application issues an MQSeries MQGET() API call to receive the message from the MQSeries queue.

Message Queuing processes the message from the initial MQSendMessage() call until it is placed on the connector queue. MSMQ-MQSeries Bridge converts and transmits the message to MQSeries, which handles the transmission from that point on.

See Also
Other Resources
How MSMQ-MQSeries Bridge Works

Sending Messages From MQSeries to Message Queuing

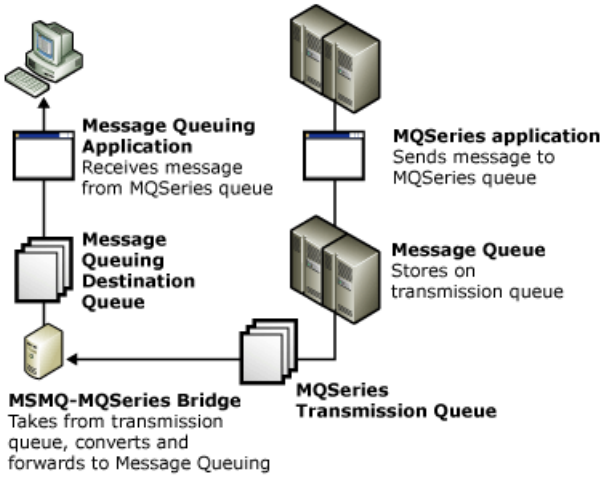
The path for sending a message from MQSeries to Message Queuing is essentially the inverse of Message Queuing to MQSeries, with a few differences. You must define appropriate MQSeries aliases, transmission queues, and channels for the Message Queuing destination queue or the Message Queuing server, and the Message Queuing destination queue must already exist.

For additional information, see Typical Configuration.

The messaging steps are as follows:

- An MQSeries application issues an MQOPEN() API call for a remote queue representing the Message Queuing destination queue.
- The MQSeries application calls MQPUT() to send a message to the remote queue. MQSeries transmits the message and stores it temporarily on an MQSeries transmission queue located at the MQSeries Queue Manager.
- MSMQ-MQSeries Bridge takes the message from the transmission queue and converts the message structure to Message Queuing message properties. MSMQ-MQSeries Bridge transmits the message to the Message Queuing destination queue.
- A Message Queuing application issues a Message Queuing MQReceiveMessage() API call to receive the message from the Message Queuing queue.

Sending a message from MQSeries to Message Queuing



See Also

Other Resources

How MSMQ-MQSeries Bridge Works

Transactional and Nontransactional Message Pipes

Using MSMQ-MQSeries Bridge, you can send messages through transactional or nontransactional message pipes.

If you send a message by a transactional message pipe, MSMQ-MQSeries Bridge supports the deliver-once feature of Message Queuing and MQSeries. This feature ensures that each message is delivered exactly once to the receiving application, but entails some overhead.

Choosing a nontransactional message pipe may improve performance, but a message may be delivered more than once in the event of a system failure during transmission.

Message Queuing messages in the foreign transactional queue will go through transactional message pipes, while Message Queuing messages in the foreign nontransactional queue will go through the nontransactional message pipes.

In the MQSeries to Message Queuing direction, you can send a message by transactional or nontransactional message pipe by specifying the appropriate remote queue manager alias as follows:

"**BRIDGEQMNAME**" refers to transactional message pipe.

"**BRIDGEQMNAME%**" refers to nontransactional message pipe.

In the Message Queuing to MQSeries direction, MSMQ-MQSeries Bridge sends transacted messages by normal service and untransacted messages by high service. In the MQSeries to Message Queuing direction, you can send a message by normal or high service by specifying an appropriate alias as the remote queue manager address.

See Also

Other Resources

How MSMQ-MQSeries Bridge Works

MSMQ-MQSeries Bridge Setup and Configuration

The following sections detail the setup and configuration of MSMQ-MQSeries Bridge.

After installing, you must configure the Message Queuing (also known as MSMQ), MQSeries, and MSMQ-MQSeries Bridge systems.

The configuration process is equivalent to building a bridge, enabling MSMQ-MQSeries Bridge to transfer messages.

In This Section

MSMQ-MQSeries Bridge Setup Requirements

The following information details the steps required to set up MSMQ-MQSeries Bridge. The installation process is straightforward, but rolling out a full installation may take some time in a complex network configuration.

This section helps you get started quickly and presents a step-by-step procedure for an initial MSMQ-MQSeries Bridge configuration.

In This Section

MSMQ-MQSeries Bridge Minimal Configuration

The minimal configuration enables MSMQ and MQSeries applications to communicate with one another, so the benefits of MSMQ-MQSeries Bridge are immediate. The minimal configuration requirements are:

- One installation of MSMQ-MQSeries Bridge on a computer running Windows Server 2003 or Windows 2000 Server.
- One installation of Message Queuing on the same computer as MSMQ-MQSeries Bridge. (Message Queuing with routing in Windows Server 2003 or Windows 2000 Server.)
- One installation of MQSeries Queue Manager on any supported platform.

See Also

Other Resources

MSMQ-MQSeries Bridge Setup Requirements

Where You Work When Installing MSMQ-MQSeries Bridge

To install MSMQ-MQSeries Bridge, you must work on two computers:

- The Message Queuing and MSMQ-MQSeries Bridge computer.
- The MQSeries computer, where the MQSeries Queue Manager is installed.

There are several steps and data transfers that must be performed on each computer.

See Also

Other Resources

MSMQ-MQSeries Bridge Setup Requirements

Gathering Required Information

Before you proceed, gather the following information. You will need this information later in the configuration process:

- Computer name (example: MSBRIDGE1).
- Directory in which the MQSeries Client for Windows Server 2003, Windows 2000, or Windows NT® is installed (example: c:\MQCLIENT).
- Connection information
    - For TCP/IP, the computer name or IP address (example: IBMNT) and the port number (example: 1414).
    - For SNA, the LU 6.2 Side Information Record (CPI-C Symbolic Destination Name).
- Name of the MQSeries Queue Manager (example: IBMNT).

Note
All MQSeries names must be in uppercase (for example, MSBRIDGE1, and not Msbridge1).

See Also

Other Resources

MSMQ-MQSeries Bridge Setup Requirements

MSMQ-MQSeries Bridge Prerequisites

For information about MSMQ-MQSeries Bridge prerequisites, see the following topics.

In This Section

MSMQ-MQSeries Bridge Platforms

Prerequisites for Computers Running Windows Server 2003 or Windows 2000

Prerequisites for MQSeries Computers

MSMQ-MQSeries Bridge Platforms

The server can be installed on any computer that is running Windows Server 2003 or Windows 2000 Server.

The Administrator Client can be installed on the following operating systems:

- Windows Server 2003
- Windows 2000 Professional
- Windows 2000 Server with Terminal Services installed

See Also

Other Resources

MSMQ-MQSeries Bridge Prerequisites

Prerequisites for Computers Running Windows Server 2003 or Windows 2000

The following software should already be installed on the computer where you will install MSMQ-MQSeries Bridge Server or Administrator Client:

Server Prerequisites

- Windows Server 2003 or Windows 2000 Server
- Message Queuing server (not in a workgroup) with routing enabled
- IBM MQSeries Client for Windows NT or IBM MQSeries for Windows NT (with server and client installed)
- TCP/IP or SNA (LU 6.2) link to an MQSeries Queue Manager (QM)

Administrator Client Prerequisites

- Windows Server 2003, Windows 2000 Professional, or Windows 2000 Server with Terminal Services
- Message Queuing set up (not in a workgroup)

See Also
Other Resources
MSMQ-MQSeries Bridge Prerequisites

Prerequisites for MQSeries Computers

The following applies to MQSeries computers:

- For OS/390 systems, be sure that your MQSeries Queue Manager is configured with the Client Attachment feature. For additional information, refer to the IBM MQSeries documentation.

See Also

Other Resources

MSMQ-MQSeries Bridge Prerequisites

MSMQ-MQSeries Bridge Properties

For information about MSMQ-MQSeries Bridge properties, see the following topics.

In This Section

Before Adding a Connected Network

How to Add a Connected Network

How to Delete a Connected Network

How to Set Connected Network Properties

How to Set Message Pipe Properties

See Also

Reference

General Tab
Advanced Tab
MQI Channels Tab
General Tab - CN
General Tab - Message Pipe
Batch Tab
Cache Tab
Retry Tab

Before Adding a Connected Network

Before you can add a connected network (CN) to an MSMQ-MQSeries Bridge, you should:

- Define the CN in Message Queuing and associate the MSMQ-MQSeries Bridge computer with the CN.
- Open MSMQ-MQSeries Bridge Manager.

See Also

Other Resources

MSMQ-MQSeries Bridge Properties

How to Add a Connected Network

To add a connected network (CN)

1. Right-click the MSMQ-MQSeries Bridge computer to which the CN is connected.
2. Select **New CN** from the pop-up menu, and select the CN name from the list.

   Along with the new CN, the following four message pipes are added to the console tree.

Message pipe	Description
MSMQ->MQS Transactional	Message Queuing to MQSeries transactional message.
MSMQ->MQS Nontransactional	Message Queuing to MQSeries nontransactional message.
MQS->MSMQ Transactional	MQSeries to Message Queuing transactional message.
MQS->MSMQ Nontransactional	MQSeries to Message Queuing nontransactional message.

3. Right-click the new CN and each message pipe to set their properties.

See Also
Other Resources
MSMQ-MQSeries Bridge Properties

How to Delete a Connected Network

To delete a connected network (CN), right-click the CN and select **Delete**.

See Also

Concepts

How to Add a Connected Network

Other Resources

MSMQ-MQSeries Bridge Properties

How to Set Connected Network Properties

To set the properties of a connected network (CN), right-click the CN icon in **MSMQ-MQSeries Bridge Manager**, and select **Properties**.

Icon	Description
	Connected network icon.

See Also

Other Resources

MSMQ-MQSeries Bridge Properties

How to Set Message Pipe Properties

To set the properties of a message pipe, right-click its icon in MSMQ-MQSeries Bridge Manager, and select **Properties**.

The following table shows the icons for message pipes.

Icon	Description
	Stopped
	Paused
	Pending
	Recovering
	Running
	Error

See Also

Other Resources

MSMQ-MQSeries Bridge Properties

Naming Message Queuing and MQSeries Entities

You must choose names for certain Message Queuing and MQSeries entities that you define during the configuration process. For more information, see the topics in this section.

Note
The names are identical to or derived from other names that you already recorded. This naming system is recommended bec ause it helps you maintain your system easily.

In This Section

Message Queuing Names

MQSeries Names

Message Queuing Names

You will need the following information when creating your Message Queuing names.

- Name of a connected network or foreign site (example: IBMNT_CN).
- Name of the Queue Manager on the foreign computer (example: IBMQM).
- Name of a foreign queue (example: IBM.NT.QUEUE).
- Name of a Message Queuing queue to which you will send test messages (example: MSBRDIGE1.QUEUE).

See Also
Concepts
MQSeries Names

MQSeries Names

You will need the following information when creating your MQSeries names.

- MQI channel name (example: IBMNT_CN).
- Transmission queue name for normal service (example: MSBRIDGE1.XMITQ).
- Transmission queue name for high service (example: MSBRIDGE1.XMITQ.HIGH).
- Name of an MQSeries queue to which you will send test messages (example: IBM.NT.QUEUE).

Installing and Configuring MSMQ-MQSeries Bridge

The following topics detail how to configure MSMQ-MQSeries Bridge components:

In This Section

How to Install the MSMQ-MQSeries Bridge Software

The procedure for installing the MSMQ-MQSeries Bridge software is the same whether you are installing the Server or the Administrator Client.

Choosing Server installs MSMQ-MQSeries Bridge and MSMQ-MQSeries Bridge Manager. Choosing Administrator Client installs only MSMQ-MQSeries Bridge Manager.

To install MSMQ-MQSeries Bridge

1. Insert the Host Integration Server CD-ROM.
2. From the Host Integration Server startup page, click **Install Server** or **Install Administrator Client** and follow the directions on the screen.
3. When the setup is complete, you can optionally view the **Readme** file and launch MSMQ-MQSeries Bridge Manager. Click **Finish** to complete the installation.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

Transport Considerations

When the TCP/IP protocol is used between the MQSeries Client and MQSeries Queue Manager, check the **Keep Alive** and **Keep Alive Time** settings in the MQSeries Queue Manager initialization file (Qm.ini). **Keep Alive** should be set to **YES** and the **Keep Alive Time** should be no more than half the message pipes retry delay. This enables the MQSeries Listener to release the resources of a broken connection before MSMQ-MQSeries Bridge retries the connection.

The message pipes retry delay can be found by right-clicking a message pipe in MSMQ-MQSeries Bridge Manager. Select **Properties**,and then select the **Cache** tab.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

Configuring MSMQ-MQSeries Bridge on Windows Server 2003 or Windows 2000

Configuring MSMQ-MQSeries Bridge on Windows Server 2003 or Windows 2000 is a three-stage process requiring Message Queuing 2.0 or later, MSMQ-MQSeries Bridge, and MQSeries to be configured in that order. Configuring Message Queuing 2.0 on Windows Server 2003 or Windows 2000 is the only stage that is different from that described for configuring MSMQ-MQSeries Bridge on Windows NT 4.0.

Message Queuing 2.0 and MSMQ-MQSeries Bridge

Message Queuing 2.0 is built into Windows Server 2003 and Windows 2000 and offers the following additional functionality:

- Integration with the Microsoft Active Directory directory service, removing the need to use a separate SQL Server™ computer to maintain the MQIS.
- Mixed-mode operation, enabling Message Queuing 1.0 and Message Queuing 2.0 environments to coexist together.
- Performance improvements, particularly in the area of transactions.
- Workgroup Mode, enabling computers running Windows Server 2003 or Windows 2000 to use Message Queuing 2.0 without the need for Active Directory.

MSMQ-MQSeries Bridge is capable of running on Windows Server 2003 or Windows 2000. MSMQ-MQSeries Bridge can be installed on any of the computers running Windows Server 2003 or Windows 2000.

To use MSMQ-MQSeries Bridge on Windows Server 2003 or Windows 2000, install:

- Windows Server 2003 or Windows 2000 Server.
- Message Queuing server (not in a workgroup) with routing enabled.
- IBM MQSeries Client for Windows NT, Version 2.0, 5.0, or 5.1, or IBM MQSeries for Windows NT, version 2.0, 5.0, or 5.1 (both Server and Client)
- MSMQ-MQSeries Bridge (from Host Integration Server)

There are some changes in Message Queuing 2.0 that are relevant to MSMQ-MQSeries Bridge. These are:

- Foreign Connected Network has been replaced with the term Foreign Site. This is in keeping with Active Directory terminology. Therefore, in Windows Server 2003 or Windows 2000 a CN refers to a foreign site.
- The Message Queuing 2.0 COM API has changed. Many more of the Message Queuing message properties are now exposed through the Message Queuing COM API. In particular, the extension property (PROPID_M_EXTENSION) is now accessible from Visual Basic®, making it easier to override MSMQ-MQSeries Bridge conversions.
- Creating foreign sites and foreign computers is achieved using Active Directory Sites and Services.

Configuring Message Queuing 2.0 on Windows Server 2003 or Windows 2000

This section assumes that Message Queuing 2.0 has been installed (with routing enabled) on a computer running Windows Server 2003 or Windows 2000 on which you plan to install MSMQ-MQSeries Bridge. Using Active Directory Users and Computers, it should be possible to see the Message Queuing object:

Domain Controllers

Computer Name (Your computer name)

Message Queuing

If this is not the case, check that the view is set to Advanced Features. If the Message Queuing object still does not appear, Message Queuing 2.0 is not installed. Check the Windows Server 2003 or Windows 2000 Message Queuing Help to ensure the Message Queuing 2.0 prerequisites are met before installing Message Queuing 2.0.

From **Control Panel**, select **Add Remove Components** and install the **Message Queuing Services**. During the installation of Message Queuing 2.0, ensure that you select **Enable Routing**. This will enable you to define a foreign site (foreign connected network) and define routing links to the foreign site, which are required to set up an MSMQ-MQSeries Bridge.

Active Directory is used to configure Message Queuing 2.0 for use with MSMQ-MQSeries Bridge. This requires permissions to make changes to Active Directory. For example, when creating a routing link, the default is that only users of the Enterprise Administrators group can make these changes. Check to ensure you have the appropriate permissions to make the required Active Directory changes before commencing Message Queuing 2.0 configuration.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Define a Foreign Site in Windows Server 2003 or Windows 2000

Use the following procedure to define a foreign site in Windows Server 2003 or Windows 2000.

To define a foreign site

1. From **Active Directory Sites and Services**, select **View**, and then select **Show Services Node**.
2. Under **Services**, right-click **MsmqServices** and select **New Foreign Site**.
3. Enter the name for the foreign site. This is the name of the Foreign Connected Network in Message Queuing 1.0. The same naming conventions apply to the foreign site name as to the Foreign Connected Network name for Message Queuing 1.0.
4. Click **OK**.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Add a Foreign Computer Representing MQSeries in Windows Server 2003 or Windows 2000

Use the following procedure to add a foreign computer representing MQSeries in Windows 2003 Server or Windows 2000.

To add a foreign computer

1. From **Active Directory Sites and Services**, select **View**, and then select **Show Services Node**.
2. Under **Services**, right-click **MsmqServices** and select **New Foreign Computer**.
3. Enter the **Name** for the Foreign Computer. This name must be the same as that of the MQSeries Queue Manager.

   Note

   Windows Server2003 and Windows2000 do not allow the dot (.) character in computer names. If your MQSeries Queue Manager name contains a dot, you can replace it with a dash (-) and MSMQ-MQSeries Bridge will map the dash back to a dot when routing the message to its MQSeries destination.

4. Select the name for the **Site**,and click **OK**. This is the name of the foreign site that was just created.
5. Click **OK**.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Set the Foreign Site Permission in Windows Server 2003 or Windows 2000

Use the following procedure to modify the permissions for the foreign site.

To set the foreign site permissions

1. From **Active Directory Sites and Services**, expand the **Sites** folder, and select the foreign site created earlier.
2. Right-click the foreign site, and then select **Properties**.
3. Select the **Security** tab, and select **Everyone**. If MSMQ-MQSeries Bridge was installed using a local account, enable **Open Connector Queue**. Otherwise, select the account name used during setup.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Create a Foreign Queue in Windows Server 2003 or Windows 2000

Use the following procedures to create a foreign queue.

To define the MQSeries queues

1. From **Active Directory Users and Computers**, select **View**, select **Advanced Features**, select **View**, and then select **Users, Groups and Computers As Containers**.
2. Expand the **Computers** folder and select the foreign computer that was previously created.
3. Right-click the Message Queuing object, select **New**, and then select **MSMQ Queue**.
4. Type the name for the queue and click **OK**.

   The name must be the same as that of the MQSeries Queue.
5. Or, if you want to create a queue to accept transactional messages, type a name, select the **Transactional** check box, and then click **OK**.

A routing link is required to enable Message Queuing 2.0 to route messages between the current Windows Server 2003 or Windows 2000 site and the newly created Foreign Site that is used for MQSeries. Use the following procedure to create a routing link.

To create a routing link

1. From **Active Directory Sites and Services**, select **View**, and then select **Show Services Node** (if not already selected).
2. Under **Services**, right-click **MsmqServices**, select **New**, and then select **MSMQ Routing Link**.
3. Set Site 1 to the name of the foreign site that was previously created. Set Site 2 to the name of the current Windows Server 2003 or Windows 2000 site.
4. Set the **Routing link cost** at 1 and then click **OK**. Using a value greater than 1 is only relevant when multiple routing links are defined between sites, and you want to enforce one route over another. Do not set this value to zero, because it will cause routing of Message Queuing messages to the foreign site to fail.

A site gate is an Message Queuing server that is configured to route messages between sites on behalf of other clients. The Message Queuing server that will be defined as the site gate is the computer that will run MSMQ-MQSeries Bridge. This site gate will use the routing link that was just created. Use the following procedure to define a site gate.

To define a site gate

1. From **Active Directory Sites and Services**, select **View**, and then select **Show Services Node** (if not already selected).
2. Under **Services**, right-click **MsmqServices**. The routing link that was previously created should appear in the right pane window.
3. **Right-click** the routing link, and select **Properties**.
4. Select the **Site Gates** tab.
5. In **Site Servers**, select the name of the computer that will run MSMQ-MQSeries Bridge, and then click **Add**.
6. Click **OK**.

The Message Queuing server, which is the same computer running Windows Server 2003 or Windows 2000 as MSMQ-MQSeries Bridge, must be added to the foreign site created earlier.

To add the site queuing server to the foreign site

1. From **Active Directory Users and Computers**, select **View**, select **Advanced Features**, select **View**, and then select **Users, Groups and Computers as containers**.
2. Expand the Domain Controllersor Computers/<*Your server name*>/MSMQ folder.

3. Right-click the **Message Queuing** object, and select **Properties**.
4. Select the **Sites** tab.
5. Select the foreign site created earlier, and then click **Add** to add this server to the foreign site.
6. Click **OK**.

On the computer running Windows Server 2003 or Windows 2000, open MSMQ-MQSeries Bridge Manager. In MSMQ-MQSeries Bridge Manager, perform the following steps.

To change the configuration in MSMQ-Series Bridge Manager

1. Expand the **Enterprise** icon.
2. Right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon, click **Stop**, and then click **OK**.
3. Right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon and click **Properties**.
4. On the **MQI Channels** tab, click **Add**.
5. Enter the **Channel Name**.
6. Enter the **Queue Manager** name.
7. For **Transport Type**, select **TCP/IP** or **SNA LU6.2**.
8. On the **Address** tab, if you connect by TCP/IP, specify the IP address or computer name for **Address** and the port number for **Port**.

   For SNA LU 6.2, specify the LU 6.2 Side Information Record.
9. Click **OK** twice.

   A message appears warning you about changing the MQI Channel configuration. Click **OK** to bypass this message.

   See the other topics in this section for more information about configuration.

See Also

Tasks

How to Export an MQSeries Server Definition File
How to Export an MQSeries Client Definition File
How to Run the MQSeries Server Definition File
How to Run the MQSeries Client Definition File

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Add the Connected Network

To add the connection network (CN), right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon in MSMQ-MQSeries Bridge Manager. Point to **New**, and click **CN**.

To add the connected network

1. Select **CN** from the drop-down list box.
2. Click **OK**. The **CN Properties** window (**General** tab) appears.
3. Select the **MQSeries QM Name**.
4. For the **Reply to QM Name**, enter the name of the computer running Windows.
5. For **Startup**, select **Enabled** and click **OK**.
6. MSMQ-MQSeries Bridge Manager displays four message pipes (`MSMQ->MQS Transactional`, `MSMQ->MQS`, and so on).

The message pipes are auto-started by default. You can disable the auto-start option in the appropriate **Properties** page.

See Also
Other Resources
Installing and Configuring MSMQ-MQSeries Bridge

How to Disable the Message Pipes

Use the following procedure to disable the message pipes.

To disable the message pipes

1. In MSMQ-MQSeries Bridge Manager, right-click the first message pipe icon, `MSMQ->MQS Transactional`, and click **Properties**.
2. On the **General** tab, select **Disabled** and click **OK**.
3. Repeat steps 1 and 2 for the second message pipe, `MSMQ->MQS`.
4. Right-click the third message pipe icon, `MQS->MSMQ Transactional`, and click **Properties**.
5. On the **General** tab, select **Disabled** and click **OK**.
6. Repeat steps 4 and 5 for the MQS->Message Queuing message pipe.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Export an MQSeries Server Definition File

Use the following procedure to export an MQSeries Server definition file.

To export an MQSeries server definition file

1. Right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon in MSMQ-MQSeries Bridge Manager, and click **Export Server Definitions**.
2. Enter a directory name to save the definition file (by default, C:\Program Files\Host Integration Server\MQBridge), and click **OK**. MSMQ-MQSeries Bridge Manager saves the file with an extension of .txt. If you define more than one **CN**, a definition file for each MQSeries Queue Manager is created in the directory.
3. Transfer the files to the MQSeries computer. If you use FTP to transfer the file, be sure to specify the ASCII transfer option.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Export an MQSeries Client Definition File

Use the following procedure to export an MQSeries Client definition file.

To export an MQSeries client definition file

1. Right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon and click **Export Client Definitions**.
2. Enter the directory name to save the definition file (the default is C:\Program Files\Host Integration Server\MQBridge), and click **OK**.
3. MSMQ-MQSeries Bridge Manager saves the file with the name ClientDf.txt.
4. Transfer the file to the MQSeries computer. If you transfer by FTP, be sure to specify the ASCII transfer option.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Run the MQSeries Server Definition File

On the MQSeries computer, run the MQSeries Server definition file that you transferred from MSMQ-MQSeries Bridge Manager.

To run the MQSeries server definition file

1. At the command prompt, run the `MQSC` command, for example:

   ```
   
   ```

   Note

   Substitute the MQSeries Queue Manager name for `IBMNT`.

2. The MQSeries Queue Manager is now configured. Review the Servreport.out file to be sure that the definitions ran successfully.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Run the MQSeries Client Definition File

On the MQSeries Queue Manager computer, run the MQSeries Client definition file that you transferred from MSMQ-MQSeries Bridge Manager.

To run the MQSeries client definition file

1. At the command prompt, run the MQSC command, for example:

   ```
   
   ```

2. The MQSC command creates a channel file called Amqclchl.tab, located in the directory MQMDirectory/QMGRS/<QueueManagerName>/@IPCC (the exact location may differ on various platforms).
3. Review the Clientreport.out file to be sure that the definitions ran successfully.
4. Transfer the Amqclchl.tab file to the MQSeries Client directory on the Windows computer. If you transfer by FTP, be sure to specify the Binary transfer option.

See Also

Other Resources

Installing and Configuring MSMQ-MQSeries Bridge

How to Configure the MQSeries Client

To complete the MQSeries configuration, return to the computer running Windows where MSMQ-MQSeries Bridge is installed.

To configure the MQSeries client

1. On the computer running Windows Server 2003 or Windows 2000, right-click **My Computer**, click **Properties**, select the **Advanced** tab, and then click **Environment Variable**.
2. Verify the following variables are in the **System Variables**.

   If they are not, set them correctly.

   MQCHLLIB should be the directory path of the channel file (by default, the MQSeries Client directory).

   MQCHLTAB should be the file name of the channel file (by default, Amqclchl.tab).
3. Check that the environment variable MQSERVER is not defined.
4. Click **OK**.

See Also
Other Resources
Installing and Configuring MSMQ-MQSeries Bridge

Testing the Installation

You can use the test programs provided in the server installation to test the MSMQ-MQSeries Bridge operation. By default, the test programs are installed to C:\Program Files\Host Integration Server\system.

For additional information about testing the MSMQ-MQSeries Bridge configuration, see the following topics.

In This Section

Creating the Test Queues

Create a Message Queuing and an MQSeries queue to receive the test messages. Alternatively, you can use queues that already exist on the computers running Windows and MQSeries.

To create a new Message Queuing queue on Windows Server 2003 or Windows 2000, see Create a Foreign Queue in Windows Server 2003 or Windows 2000.

Note
Use `IBMNT` for your MQSeries Queue Manager name. Use `IBM.NT.QUEUE` for the MQSeries queue name. **See Also**

Test the Installation

How to Test the MQSeries Client Definitions

Use the following procedure to start the MQSRRECV program.

To test the MQSeries client definitions

1. To start the program, open the command prompt and go to the MSMQ-MQSeries Bridge `samples` directory (by default, C:\Program Files\Host Integration Server\system).
2. Start the program by typing the following command:

   ```
   
   ```

   Note

   For `IBMNT`, use the MQSeries Queue Manager name. For `IBM.NT.QUEUE`, use the MQSeries queue name.

3. If the program starts successfully, it displays the message:

   ```
   Use <CTRL-C> to stop!
   ```

If you do not receive this message, confirm the channel file steps. You can view the channel file in Windows Notepad. If the file does not exist or it contains no entries, delete the Amqclchl.tab file from the MQSeries Server and execute the channel file steps again.

If the program still fails to start, override the channel file settings by issuing the following command:

```

```

An example for TCP/IP is:

```

```

An SNA example is:

```

```

Where `MQSCPIC` is the side information record name (CPI-C Symbolic Destination Name).

Then try `MQSRRECV` again. If the program now starts, either the channel file was not found, or it does not contain a correct MQI channel definition. If `MQSRRECV` still does not start, check that the listener on the MQSeries computer is started for the correct port. If not, start the listener and try again.

Press CTRL+C to stop the `MQSRRECV` program.

See Also

Other Resources

Testing the Installation

How to Start MSMQ-MQSeries Bridge

Open MSMQ-MQSeries Bridge Manager and then use the following procedure.

To start MSMQ-MQSeries bridge

1. Right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon and click **Start**.
2. In the Manager display, all four message pipes should start (the icons should have a green arrow).

If the message pipes do not start, check the event log.

The `MQS->MSMQ` message pipes may fail in the following situations:

- Another MSMQ-MQSeries Bridge computer is currently using the same transmission queue.
- Microsoft MSMQ-MQSeries Bridge is terminated abnormally, causing the MQSeries server to behave as if MSMQ-MQSeries Bridge is still using the corresponding XMIT queue. If this happens, you must restart MQSeries Queue Manager, or configure the **Keep Alive** option in your MQSeries server. (For further information, see your MQSeries documentation.)

See Also
Other Resources
Testing the Installation

How to Send Test Messages from Message Queuing to MQSeries

On the computer running Windows, perform the following procedure to send test messages.

To send test messages from message queuing to MQSeries

1. Open two MS-DOS windows. Change both windows to the server installation directory (by default, C:\Program Files\Host Integration Server\System).
2. Start the MQSeries receiver program in the first window.

Note
Use `IBMNT` for your MQSeries Queue Manager name. Use `IBMNT` for the MQSeries queue name.

3. In the second window, send 10 messages from Message Queuing to MQSeries:

Note
Use `IBMNT` for your foreign computer name. Use `IBMNT.QUEUE` for the MQSeries queue name.

4. If the test succeeds, the first window displays the 10 messages that it receives.
5. Stop the `MQSRRECV` program by pressing CTRL+C in the first window.

See Also

Other Resources

Testing the Installation

How to Send Test Messages from MQSeries to Message Queuing

On the computer running Windows, perform the following procedure to send test messages.

To send test messages from MQSeries to Message Queuing

1. Open two MS-DOS windows. Go to the default installation directory (default is C:\Program Files\Host Integration Server\System).
2. In the first window, start the Message Queuing receiver program.
3. Send from MQSeries with the following command:

```
```

4. Receive in Message Queuing with the following command:

```
    >MSMQRecv MSBRIDGE1\MSBRIDGE1.QUEUE
```

See Also

Other Resources

Testing the Installation

Typical Configuration

The topics in this section detail the configuration of MSMQ-MQSeries Bridge.

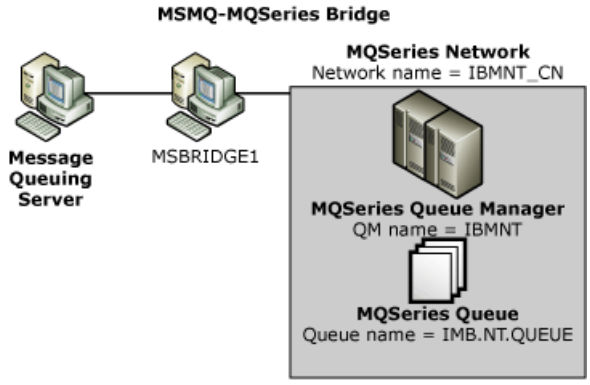
In This Section

Typical Configuration Diagram

Examine the diagram for an overview of a typical configuration. The names in the diagram (such as MSBRIDGE1, IBMNT_CN, and IBMNT) are arbitrary, but you must use the same name for the same entity at different locations in the configuration.

For example, if the MSMQ-MQSeries Bridge computer is called MSBRIDGE1, you must reference this same name in MSMQ-MQSeries Bridge Manager and in MQSeries.

A typical configuration



Settings for a Typical Configuration

Typical Configuration Settings

The following tables show settings for a typical configuration.

Define in Message Queuing		
	Foreign Connected Network or Foreign Site	IBMNT_CN
	Foreign computer	IBMNT
	Foreign queue	IBM.NT.QUEUE

Define in MSMQ-MQSeries Bridge		
MSMQ-MQSeries Bridge properties	MQI channel	IBMNT_CN
	Transport type	TCP/IP or LU6.2
Foreign Connected Network or Foreign Site properties	CN name	IBMNT_CN
	MQSeries QM	IBMNT
	Reply to QM	MSBRIDGE1
Message pipe properties	MQSeries Message Queuing	Normal
	MQSeries Message Queuing	High
Transmission queue	IBM.NT_CN.XMITQ	IBMNT_CN.XMITQ.HIGH

Import or define in MQSeries		
	Transactional Service	Nontransactional Service
Transmission queue	IBMNT_CN.XMITQ	IBMNT_CN.XMITQ.HIGH
Queue manager alias	MSBRIDGE1	MSBRIDGE1%
	Model queue	Q2Q_SYNC_Q
	MQI channel	MQS_CN
	Transport type	TCP/IP or LU6.2

See Also

Concepts

Typical Configuration Diagram

MSMQ-MQSeries Bridge Manager

For information about MSMQ-MQSeries Bridge Manager, see the following topics.

In This Section

MSMQ-MQSeries Bridge Manager Overview

Using MSMQ-MQSeries Bridge Manager, you can control and monitor MSMQ-MQSeries Bridge operations. For example, you can use MSMQ-MQSeries Bridge Manager to:

- Start, stop, or pause the operation of an MSMQ-MQSeries Bridge, a CN, or a message pipe.
- Observe the quantity and type of traffic between Message Queuing and MQSeries.
- Determine how many messages are on MQSeries transmission queues or Message Queuing connector queues waiting to be transmitted.

This topic explains how to use and customize MSMQ-MQSeries Bridge Manager for these purposes.

Note
You can also use MSMQ-MQSeries Bridge Manager to configure your MSMQ-MQSeries Bridge system. This is an essential st ep in the MSMQ-MQSeries Bridge installation. For complete information, see MSMQ-MQSeries Bridge configuration.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

MSMQ-MQSeries Bridge Manager Display

Open MSMQ-MQSeries Bridge Manager. The MSMQ-MQSeries Bridge Manager window is divided into two panes. On the left, in the console tree, a Manager tree displays the MSMQ-MQSeries Bridge installations in your network, and message pipes. The details pane, on the right, lists detailed information about the status of an object that is selected in the tree.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

MSMQ-MQSeries Bridge Manager Properties

To set the properties of MSMQ-MQSeries Bridge components, right-click the **Microsoft MSMQ-MQSeries Bridge Service** icon from the left pane in MSMQ-MQSeries Bridge Manager, and select the **Properties** option.

The Properties window displays several tabs. You should set the properties on the **MQI Channels** tab before you add a connected network.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Icons for MSMQ-MQSeries Bridge Manager Objects

In MSMQ-MQSeries Bridge Manager, icons are used to designate items such as MSMQ-MQSeries Bridge installations, connected networks (CNs), and message pipes.

Icon	Description
	Enterprise network
	Computer (on which one or more MSMQ-MQSeries Bridge products are installed)
	MSMQ-MQSeries Bridge (running)
	MSMQ-MQSeries Bridge (stopped)
	Connected Network
	Message pipe (running)
	Message pipe (paused)
	Message pipe (pending)
	Message pipe (recovering)
	Message pipe (stopped)
	Message pipe (error)

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Column Display Options for MSMQ-MQSeries Bridge Manager

The columns displayed in the details pane of MSMQ-MQSeries Bridge Manager change according to the object you have selected in the console tree. You can customize the columns that MSMQ-MQSeries Bridge Manager displays for each type of object.

Note
The Object Name column is always displayed.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

MSMQ-MQSeries Bridge Display

The following table shows the MSMQ-MQSeries Bridge Manager column name and description.

Column name	Description
MQS H Threads	Number of MQSeries to Message Queuing threads at nontransactional message pipe
MQS N Threads	Number of MQSeries to Message Queuing threads at transactional message pipe
Message Queuing H Threads	Number of Message Queuing to MQSeries threads at nontransactional message pipe
Message Queuing N Threads	Number of Message Queuing to MQSeries threads at transactional message pipe
Lifetime	Time since MSMQ-MQSeries Bridge started
Path	Computer name
Status	MSMQ-MQSeries Bridge status (running, paused, or stopped)
DLQ Depth	Number of messages on the MSMQ-MQSeries Bridge non-transacted dead letter queue
XDLQ Depth	Number of messages on the MSMQ-MQSeries Bridge transacted dead letter queue

See Also
Other Resources
MSMQ-MQSeries Bridge Manager

Connected Network Display

The following table shows the connected network (CN) column name and description.

Column name	Description
QM Name	Name of MQSeries Queue Manager to which the CN is connected
Startup	CN is enabled or disabled at MSMQ-MQSeries Bridge startup
Status	CN status (running, paused or stopped)

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Message Pipe Display

The following table shows the message pipe column name and description.

Column name	Description
Acc Size	Accumulated size (in bytes) of all messages transmitted since the message pipe started
Lifetime	Time since the message pipe started
Messages Sent	Total number of messages that have been sent since the message pipe started
Msgs/Sec	Messages per second (the current throughput of the pipe)
Q Depth	Queue depth (number of messages in the queue) waiting to be transmitted
Q Name	Name of the MQSeries transmission queue or Message Queuing connector queue associated with the message pipe
Retries	Number of times that MSMQ-MQSeries Bridge has tried to activate the message pipe
Startup	Whether the message pipe is enabled or disabled at MSMQ-MQSeries Bridge startup
Status	Message pipe status (running, paused, pending, recovering, stopped, or error)

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

How to Customize the Column Display

Use the following procedure to add or remove a column in the details pane of MSMQ-MQSeries Bridge Manager.

To customize the column display

1. From the **View** menu, select **Columns**.
2. Select the tab for the type of object (**MSMQ-MQSeries Bridge**, **CN**,or **Message Pipe**).
3. To add a column to the display, select its name in the **Available Columns** list, and click **Add**.
4. To remove a column, select its name in the **Show the following** list, and click **Remove**.
5. To change the sequence of the column display, select columns in the **Show the following** list, and click the UP ARROW key or the DOWN ARROW key.
6. Click **OK**.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Shortcut Menu in MSMQ-MQSeries Bridge Manager

MSMQ-MQSeries Bridge Manager offers the standard Windows Explorer menu and toolbar. In addition, you can display a shortcut menu by right-clicking:

New

Start

Stop

Refresh Cache

Pause

Resume

Delete

Export Server Definitions

Export Client Definitions

Properties

The options on the menu depend on the object and its current status.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Status Bar

The status bar at the bottom of the MSMQ-MQSeries Bridge Manager window displays status and error messages. For example, the status bar displays an access denied message if an unauthorized user tries to start or stop Microsoft MSMQ-MQSeries Bridge.

See Also

Other Resources

MSMQ-MQSeries Bridge Manager

Controlling MSMQ-MQSeries Bridge

To control the operation of an object, right-click the object in MSMQ-MQSeries Bridge Manager, and select the appropriate command. Some of the commands are also available on the main **Manager** menu.

For additional information, see the following topics.

In This Section

Starting, Stopping, or Pausing an Object

At MSMQ-MQSeries Bridge startup, the connected network (CN) and message pipes start automatically if you selected the **Startup Enabled** option in the object properties.

To change the status of an object, right-click that object in MSMQ-MQSeries Bridge Manager. Select one of the following options.

Option	Description
Stop	Stops the operation and resets the object's counters (lifetime, messages sent, and accumulated size).
Start	Starts a stopped object.
Pause	Pauses an operation, retaining the object's counters.
Resume	Resumes operation after **Pause**.

The command affects the selected object and all objects below it on the tree. For example, starting a CN starts all its **Enabled** message pipes.

See Also

Other Resources

Controlling MSMQ-MQSeries Bridge

Refreshing the Cache

Occasionally, you may want to refresh the MSMQ-MQSeries Bridge cache memory. For example, you can do this to close queues needed by other applications.

In MSMQ-MQSeries Bridge Manager, right-click a Microsoft MSMQ-MQSeries Bridge service or message pipe, and choose the **Refresh Cache** option.

See Also

Other Resources

Controlling MSMQ-MQSeries Bridge

Configuration Backup

MSMQ-MQSeries Bridge stores the MQSeries configuration and the MSMQ-MQSeries Bridge configuration in the registry. You can back up those configurations by using Regedit.exe provided in your Windows Server 2003, Windows 2000, or Windows NT system.

The registry location depends on whether you are running Windows NT in a cluster or non-cluster environment, as follows.

MSMQ-MQSeries configuration	Registry key to backup
Non-cluster environment	HKLM\Software\Microsoft\MQBridge\Server
Cluster environment	HKLM\Cluster\Software\Microsoft\MQBridge\Server

See Also

Other Resources

Controlling MSMQ-MQSeries Bridge

Additional Information About MSMQ-MQSeries Bridge

The following topics provide additional information about MSMQ-MQSeries Bridge.

In This Section

Using a Dash (-) in Message Queuing Computer Names

If a Message Queuing computer name contains the dash (-) character, IBM MQSeries will not be able to address it as a remote queue manager because IBM MQSeries does not support queue manager names with dashes.

MSMQ-MQSeries Bridge provides a feature to resolve this problem through name replacement. In IBM MQSeries, you can refer to the remote queue manager name with a dot (.) instead of a dash, and when the message arrives, MSMQ-MQSeries Bridge changes the dot back to a dash.

To enable this option, see the Advanced Tab.

See Also

Other Resources

Additional Information About MSMQ-MQSeries Bridge

Using a Dot (.) in Remote MQSeries Queue Manager Names on Windows Server 2003 or Windows 2000

Windows Server 2003 or Windows 2000 does not support computer names containing the dot (.) character. As a result, dots cannot be used when naming a remote IBM MQSeries QM.

MSMQ-MQSeries Bridge provides a feature to resolve this problem through name replacement. In Windows Server 2003 or Windows 2000, you can refer to the remote IBM MQSeries QM name with a dash (-) instead of a dot, and when the message arrives, MSMQ-MQSeries Bridge changes the dash back to a dot.

This feature is enabled by default. To disable the feature, set the following registry value:

Key: HKLM\Software\Microsoft\MQBridge\Server

Value: DisableDash2DotTranslation(REG_DWORD) = 1

See Also

Other Resources

Additional Information About MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Dead Letter Queue

MSMQ-MQSeries Bridge creates two dead letter queues for storing messages that are not deliverable. **MQBridge Dead Letter Queue** is for messages through the nontransactional message pipe, and **MQBridge Xact Dead Letter Queue** is for messages through the transactional message pipe.

In Windows Server 2003 or Windows 2000, these two queues are located in **Active Directory Users and Computers** under **Domain Controllers** or **Computers\<MSMQ-MQSeries Bridge Computer>\msmq**.

See Also

Other Resources

Additional Information About MSMQ-MQSeries Bridge

Notes on the Current Release of MSMQ-MQSeries Bridge

The following are additional notes to support the current release of MSMQ-MQSeries Bridge in Host Integration Server.

MSMQ-MQSeries Bridge Manager Supports Queue Names in Uppercase Characters

MSMQ-MQSeries Bridge does not differentiate uppercase vs. lowercase foreign queue names and foreign queue manager names. It converts them all to uppercase. MSMQ-MQSeries Bridge can only send messages to MQSeries queue managers and queues that have uppercase names.

MSMQ-MQSeries Bridge Encryption Delays on Windows Server 2003 or Windows 2000

There is a noticeable delay for changes to take place when enabling or disabling MSMQ-MQSeries Bridge encryption. It is recommended that you wait 5 to 15 minutes after changing the **Enable/Disable** option for encryption, before expecting MSMQ-MQSeries Bridge to function correctly.

Converting an MSMQ-MQSeries Bridge Resource Into a Cluster Resource

In Windows Server 2003 or Windows 2000, clusters are Active-Active. You must perform the following steps after installation to convert an MSMQ-MQSeries Bridge resource into a cluster resource:

1. In the Program Files\Host Integration Server\System directory by default, at the command line, type:

   ```
   BCLUSTER
   ```

2. When the dialog box appears, click **Add Bridge Resource**.

3. In MSMQ-MQSeries Bridge Manager, right-click the **Microsoft MSMQ-MQSeries Bridge Service** cluster resource and select **Bring online**.

In Windows Server 2003 or Windows 2000, you can also run the cluster on the local node only. To do this, run BCLUSTER as described in the preceding procedure, and click **Remove Resource**.

When moving from local node to cluster node, it is necessary to stop Microsoft MSMQ-MQSeries Bridge on all local nodes before you can bring the Microsoft MSMQ-MQSeries Bridge cluster resource online. This restriction only applies when running on Windows Server 2003 or Windows 2000.

Granting Access for Users Created During Install

When installing MSMQ-MQSeries Bridge on a computer running Windows Server 2003 or Windows 2000, if you choose to automatically create a user, that user will be denied access when attempting to start MSMQ-MQSeries Bridge.

You may also receive the following error in the event log: "Access to MSMQ connector queue is denied." While you should have sufficient rights for the operation of Microsoft MSMQ-MQSeries Bridge, it may be necessary to manually configure the Domain Administrators group security for the foreign site or sites to have full control.

To grant user access

1. In **Active Directory**, go to **Active Directory Sites and Services**.
2. Right-click the foreign site.
3. Click the **Security** tab, and click **Domain Admins**.
4. In the **Permissions** box, select the **Full Control Allow** box.
5. Click **OK**.

MSMQ-MQSeries Bridge Quick Setup for Windows Server 2003 or Windows 2000

Use the following steps to set up a foreign site in Windows Server 2003 or Windows 2000 (previously known as a foreign connected network in Windows NT 4.0).

1. Install Message Queuing with routing.

2. Use **Active Directory Sites and Service** to create a new foreign site and new foreign computer. It is located in the Services\MsmqServices folder (right-click).

3. Create a routing link between the foreign site and the default first site name by right-clicking the Services\MsmqServices folder. Click **New**, and then click **MSMQ Routing Link**.

4. In **Routing link cost**, enter a number between 1-999, where 1 has the highest priority. Do not enter 0; it means no link.

5. Select **Property**, choose the **Site Gates** tab, and then add the MSMQ-MQSeries Bridge computer to be a member of the routing link site gates.

6. In **Active Directory Users and Computers**, under **Domain Controllers** or **Computers\<MSMQ-MQSeries Bridge Computer>\msmq**, select **Property** and then select the **Sites** tab. Add the computer to the foreign site created earlier.

7. Recycle Message Queuing on all domain controllers.

Testing the MQSeries Client/Server Connection Before Using Message Queuing Applications

The connection between the MQSeries Client and the MQSeries Server should be tested using the AMQSPUTC and AMQSGETC tools from IBM. To perform this test, use the following procedure.

1. Configure the MQSeries Server by using the server and client definition files generated by MSMQ-MQSeries Bridge.

2. Place the .tab file (by default, Amqclchl.tab) in the client location.

3. Use AMQSPUTC and AMQSGETC to test the connection.

Enabling MSMQ-MQSeries Bridge Encryption

Before you can enable encryption, you must first run the Add Schema program.

1. Log on as Schema Administrator. (This is usually the same as a Domain Administrator.)

2. In the Program Files\Host Integration Server\System directory (by default), type the following at the command line:

```
Addschma hiserver.schema
```

3. In MSMQ-MQSeries Bridge Manager, expand **Enterprise**, expand **Computers**, and expand the computer name.

4. Right-click **Microsoft MSMQ-MQSeries Bridge Service**, click **Properties**, and then click the **Advanced** tab.

5. Select the **Support MSMQ to Bridge Encryption** box.

6. Click **OK**, click **Yes**, and then click **OK**.

7. Restart the Microsoft MSMQ-MQSeries Bridge.

To disable encryption, return to the **Advanced** tab and clear the **Support MSMQ to Bridge Encryption** box, and then recycle Microsoft MSMQ-MQSeries Bridge.

MSMQ-MQSeries Bridge Rejects MSMQ Messages with MQMSG_AUTH_LEVEL_MSMQ20 Authentication

Windows Server 2003 or Windows 2000 Server Message Queuing 2.0 introduces a new authentication signature,

MQMSG_AUTH_LEVEL_MSMQ20, and is not supported for a connector queue. The result is that messages are rejected without being processed.

See Also

Other Resources

Additional Information About MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Terminology

The following terms are used in MSMQ-MQSeries Bridge.

Term	Definition
CN	Foreign Connected Network in Windows NT 4.0, and Foreign Site in Windows Server 2003 or Windows 2000.
DLQ and **XDLQ**	MQBridge Dead Letter Queue and MQBridge Xact Dead Letter Queue. (For more information, see MSMQ-MQSeries Bridge Dead Letter Queue.)
MCA	Message Channel Agent. (For details, see IBM MQSeries documentation.)
MQI	Message Queue Interface (For details, see IBM MQSeries documentation.)
MQS	IBM MQSeries.
Message Queuing	Also known as MSMQ. Formerly known as Microsoft Message Queuing, Message Queuing Services, or Message Queue Server.
Nontransactional Message Pipe	Option allowing possible duplicate delivery of a message.
QM	Queue Manager.
Transactional Message Pipe	Option guaranteeing delivery of a message one time and only one time.

See Also
Other Resources
Additional Information About MSMQ-MQSeries Bridge

Security User's Guide

Enterprise Single Sign-On (SSO) provides services to enable single sign-on for end users in enterprise application integration (EAI) solutions. The SSO system maps Microsoft Windows accounts to back-end credentials. SSO simplifies the management of user IDs and passwords, both for users and administrators. It enables users to access back-end systems and applications by logging on only one time to the Windows network.

Note
Performance monitoring is available in this release. If you are running Enterprise SSO on a 64-bit Windows computer, you must also use the 64-bit Perfmon tool in order to use the ESSO Perfmon counters.

In This Section

See Also

Other Resources

Operations

Understanding Enterprise Single Sign-On

To understand Enterprise Single Sign-On (SSO), it is useful to look at the three types of Single Sign-On services available today: Windows integrated, extranet, and intranet. These are described in the following sections, with Enterprise Single Sign-On falling into the third category.

Windows Integrated Single Sign-On

These services enable you to connect to multiple applications within your network that use a common authentication mechanism. These services request and verify your credentials after you log into the network, and use your credentials to determine the actions that you can perform based on your user rights. For example, if applications integrate using Kerberos, after the system authenticates your user credentials, you can access any resource in the network that is integrated with Kerberos.

Extranet Single Sign-On (Web SSO)

These services enable you to access resources over the Internet by using a single set of user credentials. The user provides a set of credentials to log on to different Web sites that belong to different organizations. An example of this type of Single Sign-On is the Microsoft Passport Network for consumer-based applications. For federated scenarios, Active Directory Federation Services enables Web SSO.

Server-Based Intranet Single Sign-On

These services enable you to integrate multiple heterogeneous applications and systems in the enterprise environment. These applications and systems might not use common authentication. Each application has its own user directory store. For example, in an organization, Windows uses Active Directory directory service to authenticate users, and mainframes use IBM's Resource Access Control Facility (RACF) to authenticate the same users. Within the enterprise, middleware applications integrate the front-end and back-end applications. Enterprise Single Sign-On enables users in the enterprise to connect to both the front end and back end while using only one set of credentials. It enables both Windows Initiated Single Sign-On (in which the initial request is made from the Windows domain environment) and Host Initiated Single Sign-On (in which the initial request is made from a non-Windows domain environment) to access a resource in the Windows domain.

In addition, Password Synchronization simplifies administration of the SSO database, and keeps passwords in sync across user directories. You can do this by using password synchronization adapters, which you can configure and manage using the Password Synchronization tools.

Enterprise Single Sign-On System

Enterprise Single Sign-On provides services to store and transmit encrypted user credentials across local and network boundaries, including domain boundaries. SSO stores the credentials in the Credential database. Because SSO provides a generic single sign-on solution, middleware applications and custom adapters can take advantage of SSO to securely store and transmit user credentials across the environment. End users do not have to remember different credentials for different applications.

SSO System Components

The Single Sign-On system consists of a Credential database, a master secret server, and one or more Single Sign-On servers.

The SSO system contains affiliate applications that an administrator defines. An affiliate application is a logical entity that represents a system or sub-system such as a host, back-end system, or line-of-business application to which you are connecting using Enterprise Single Sign-On. Each affiliate application has multiple user mappings; for example, it has the mappings between the credentials for a user in Active Directory and their corresponding RACF credentials.

The Credential database is the SQL Server database that stores the information about the affiliate applications, as well as all the encrypted user credentials to all the affiliate applications.

The master secret server is the Enterprise Single Sign-On server that stores the master secret. All other Single Sign-On servers in the system obtain the master secret from the master secret server.

The SSO system also contains one or more SSO servers. These servers do the mapping between the Windows and back-end credentials and look up the credentials in the Credential database. Administrators use them to maintain the SSO system.

Note
You can have only one master secret server and only one Credential database in your SSO system. The Credential database c an be remote to the master secret server.

Note

Enterprise Single Sign-On has limited functionality in a workgroup environment, supporting only config store scenarios. A domain environment is required for Single Sign-On scenarios, Password Sync scenarios, and ESSO Management Agent scenarios with MIIS.

In This Section

- Enterprise Single Sign-On User Groups
- SSO Components
- SSO Server
- Master Secret Server
- SSO Affiliate Applications
- SSO Mappings
- SSO Tickets
- Configuring Enterprise Single Sign-On

Enterprise Single Sign-On User Groups

To configure and manage the Enterprise Single Sign-On (SSO) system, you must create certain Windows groups and accounts for each of these roles. When configuring the access accounts in Enterprise SSO, you can specify more than one account for each of these roles. This section describes these roles.

Important
It is strongly recommended that you use domain groups when configuring SSO.

Note
For security purposes, the SSO system does not allow for built-in accounts.

Single Sign-On Administrators

SSO administrators have the highest level user rights in the SSO system. They can do the following:

- Create and manage the Credential database.
- Create and manage the master secret.
- Enable and disable the SSO system.
- Create password synchronization adapters.
- Enable and disable password synchronization in the SSO system.
- Enable and disable host-initiated SSO.
- Perform all administration tasks.

The SSO administrators account can be either a Windows group account or an individual account. The SSO administrators account can also be either a domain or local group or individual account. When you use an individual account, you cannot change it to another individual account. Therefore, it is recommended that you do not use an individual account. You can change this account to a group account as long as the original account is a member of the new group.

Important
The service account that runs the Enterprise Single Sign-On service must be a member of this group. To help secure your env ironment, ensure that no other service is using the same service account.

Single Sign-On Affiliate Administrators

The SSO affiliate administrator defines the affiliate applications that the SSO system contains. Affiliate applications are a logical entity that represents the back-end system to which you are connecting using SSO. SSO affiliate administrators can do the following:

- Create and manage affiliate applications.
- Specify the application administrators account for each affiliate application.
- Perform all the administration tasks that the application administrators and application users can.

The SSO Affiliate Administrator account can be either a Windows group account or an individual account. The SSO Affiliate Administrator account can also be either a domain or local group or account.

Application Administrators

There is one application administrators group per affiliate application.

Members of this group can do the following:

- Change the application users group account.
- Create, delete, and manage credential mappings for all users of the specific affiliate application.
- Set credentials for any user in that specific affiliate application users group account.
- Perform all the administration tasks that the application users can.

Application Users

There is one application users group account for each affiliate application. This group contains the list of end users in an Enterprise SSO environment. Members of this group can do the following:

- Look up their credentials in the affiliate application.
- Manage their credential mappings in the affiliate application.

Note
Remember to be vigilant when assigning groups. It is possible, for example, to use a Host Integration Server security user group for the SSO application users group. Before you do this, be certain that all users need all access that will then be available to them.

See Also

Tasks

How to Update the Properties of an Affiliate Application

How to Update the Credential Database

Concepts

Understanding Enterprise Single Sign-On

Other Resources

Managing User Mappings

SSO Components

The sub services of the Enterprise Single Sign-On (SSO) service are as follows:

- **Mapping.** Maps the user account in the Windows system to the user accounts in the back-end systems (affiliate applications).
- **Lookup.** Looks up the user credentials in the Credential database in the back-end system. This is the SSO runtime component.
- **Administration.** Manages the affiliate applications and the mappings for each affiliate application.
- **Secret.** Generates the master secret and distributes it to the other SSO servers in the system. It is only active on the Single Sign-On server that is acting as the master secret server.
- **Password Synchronization.** Simplifies administration of the SSO credential database, and keeps passwords in sync across user directories.

See Also
Concepts
Understanding Enterprise Single Sign-On
SSO Server
Other Resources
Installing Enterprise Single Sign-On

SSO Server

The Enterprise Single Sign-On (SSO) server can perform any of the following tasks:

- **Functions as the master secret server.** The master secret server holds the master secret, or encryption key, used to encrypt all the credentials in the SSO system. Though the master secret server can act as a server for lookups and administration, it is recommended that you use this server to act only as a master secret server for security reasons. For more information about the functions the master secret server performs, see Master Secret Server.

- **Performs administrative operations.** SSO administrators can use any of the Single Sign-On Servers to perform administrative tasks such as managing affiliate applications, setting user credentials, and managing user mappings.

- **Performs lookup operations.** The SSO server uses the runtime component to look up the user credentials.

- **Issues and Redeems Tickets.** The SSO server also issues and redeems SSO tickets, which applications can use to get user credentials.

- **Password Synchronization.** You can create and manage password synchronization adapters on the SSO Server.

See Also
Concepts
SSO Tickets
Master Secret Server
Other Resources
Using Enterprise Single Sign-On
Installing Enterprise Single Sign-On
Password Synchronization

Master Secret Server

The *master secret server* is the Enterprise Single Sign-On (SSO) server that stores the master secret (encryption key). The master secret server generates the master secret when an SSO administrator requests it. The master secret server stores the encrypted master secret in the registry. Only Single Sign-On administrators can access the master secret.

The other Single Sign-On servers check every 30 seconds to see whether the master secret has changed. If it has changed, they read it securely; otherwise, they continue to use the master secret they already have cached in memory. The SSO service uses the master secret to encrypt and decrypt the user credentials.

You cannot use the SSO system until an SSO administrator configures the master secret server and generates the master secret. The master secret server generates the master secret during configuration. Only SSO administrators can generate the master secret. An SSO administrator must configure the master secret server and the Credential database before an application can use the SSO service.

Important
After you generate the master secret, it is strongly recommended that you back it up and store it in a secure location.

When an SSO administrator needs to regenerate the master secret, for example, if the SSO administrator wants to change the master secret periodically, the master secret server stores both the old and new master secret. The master secret server then goes through all the mappings, decrypts them using the old master secret, and encrypts them again using the new master secret.

If the master secret server fails, all runtime operations that are already running continue to run, but SSO servers are not able to encrypt new credentials.

Important
There can be only one master secret server in your SSO system. Therefore, it is strongly recommended you cluster the maste r secret server. For more information, see How to Cluster the Master Secret Server.

See Also

Other Resources

Using Enterprise Single Sign-On

Managing the Master Secret

SSO Affiliate Applications

The Enterprise Single Sign-On (SSO) affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line-of-business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

When the SSO administrator or the SSO affiliate administrator defines an affiliate application, they must also determine who will administer the affiliate application (the application administrator), who the users of the affiliate application are (the application users), and what parameters the SSO system will use to authenticate the users of this affiliate application (the user ID, passwords, PINs, and so on). For more information about application administrators and application users, see Enterprise Single Sign-On User Groups.

Affiliate Application Types

Enterprise SSO defines several different application types. The different application types support different types of mappings between the Windows account and the account on the non-Windows system.

The application types are as follows:

Individual Individual applications support one-to-one mappings between the Windows account and the non-Windows account. In an Individual type application, one Windows account is mapped to one, and only one, non-Windows account. The mapping can be used in either direction, from Windows to non-Windows, or from non-Windows to Windows, or both, depending on the flags that have been set for this application. Thus, Individual applications can be used for Windows-initiated SSO, Host-initiated SSO, or both.

Group Group applications support mappings between one Windows group to one single non-Windows account. The Application Users account is used to define the Windows group that will be used for this Group application. Only one mapping can be defined for a Group application, and that mapping must be between the Windows group and the single non-Windows account that will be used by all members of this Windows group to access the non-Windows system. Group applications can only be used for Windows initiated SSO.

Host Group Host Group applications are conceptually the reverse of Group applications. They support mappings between a defined group of non-Windows accounts to a single Windows account. The single Windows account that will be used by the non-Windows accounts is defined by the Application Users account for the application. The group of non-Windows accounts that is allowed to access this application is defined by creating a mapping for each non-Windows account. Host Group applications can only be used for Host initiated SSO.

Designing an Affiliate Application

Before creating an affiliate application, the SSO affiliate administrator or the SSO administrator must make the following decisions:

1. **What will this affiliate application represent?** You must know the non-Windows application that the affiliate application will represent in the SSO system. For example:

   Application name: APP1

   Description: Application for Pay stub department

   Contact: administrator@companyname.com

2. **Who will administer this affiliate application?** You must determine who the administrators are for this affiliate application. These form the Windows administrators group for this affiliate application; for example, Domain\APP1AdminGroup.

3. **Who will use this affiliate application?** You must determine who the end users are for this affiliate application. These users represent the Windows users group for this affiliate application; for example, Domain\DomainUsers. In the case of the application for Pay stubs, you might want all users to access their pay stub information, so you can specify the domain users group as the user group for this application.

4. **What credentials does the affiliate application use to authenticate its users?** Different applications use different credentials to authenticate users. For example, some applications might use user IDs, passwords, PINs, or a combination

of these. You must also determine whether the system needs to mask these credentials as the user provides them.

5. **Will you use individual mappings or a group mapping for this affiliate application?** Does each Windows user have an account in the back-end system, or does the back-end system have one account for all Windows users? In the case of the pay stub system, each user has an account to access individual pay stub information, and you would need to use individual mappings.

After you create an affiliate application, you cannot modify the following properties:

- Name of the affiliate application.
- Fields associated with the affiliate application.
- Affiliate application type (host group, individual, or configuration store).
- Administration account same as affiliate administrators group. (If you select this property, the affiliate administrators group is used as the application administrators account for this affiliate application.)

Affiliate Application Properties

The following table lists the properties that you must define for each affiliate application that you create.

Property	Description
Application name	Name of the affiliate application. You cannot change this property after you create the affiliate application.
Description	Brief description of the affiliate application.
Contact	The main contact for this affiliate application that users can use. (Can be an e-mail address.)
appUserAccount	The Windows group that contains the user accounts of end users who will use this affiliate application.
appAdminAccount	The Windows group that contains the administrator accounts that will manage this affiliate application. Note You do not need to define this property if you set the adminAccountSame to Yes.

Application Flag	Description
enableApp	The status of this affiliate application.
groupApp	Determines whether this application uses a group mapping (Yes) or individual mappings (No). You cannot change this property after you create the application.
configStoreApp	Determines whether this affiliate application is a Configuration Store type application (Yes). You cannot change this property after you create the application.
hostInitiatedSSO	Enable this if it is a host-initiated SSO type application. Default is No.
windowsInitiatedSSO	Enable this if it is a Windows initiated SSO type application. Default is Yes.
validatePassword	This applies only to host-initiated SSO applications. When the application tries to retrieve credentials, it must provide the password in the Credential database, which is used for validation by SSO services. Default is Yes.

disableCredCache	The SSO server stores credentials in a cache to expedite access. Default is No.
allowTickets	Determines whether the SSO system uses tickets for this affiliate application. Note You must be an SSO administrator to set this flag.
validateTickets	Determines whether the SSO system validates tickets when the user redeems them. Note You must be an SSO administrator to set this flag.
appTicketTimeOut	Specifies a ticket time-out specific to the affiliate application. You can set this only when updating an affiliate application, not when creating it. If ticketing is enabled for this application and this property is not, the time-out specified at the SSO System (Global) level is used. Note You must be an SSO administrator to set this flag.
timeoutTickets	Determines whether tickets have an expiration time. Default is No. Note You must be an SSO administrator to set this flag.
allowLocalAccounts	Determines whether you allow the use of local groups and accounts in the SSO system. You can only configure this flag to Yes in single-computer scenarios.
adminAccountSame	Determines whether to use the SSO affiliate administrator group as the application administrator group. You cannot change this property after you create the application. Note You must be an SSO administrator to set this flag.

Application Fields	**Description**	**Description**
Field [0]	<*credential*>: Masked/Unmasked	Determines the type of credential (user ID, password, smartcard) that end users must provide to connect to the affiliate application, and whether this credential is masked (that is, whether the characters that the user types are displayed on the screen). You can enter as many fields as there are credentials for the affiliate application, but the first field must be the user ID. You cannot change this property after you create the application.

See Also

Concepts

SSO Mappings

Understanding Enterprise Single Sign-On

Other Resources

Managing Affiliate Applications

Managing User Mappings

SSO Mappings

When an Enterprise Single Sign-On (SSO) administrator or an SSO affiliate administrator defines an affiliate application, the administrator can define it either as an application that uses individual mappings, or as an application that uses a group mapping.

Individual Mappings

SSO individual mappings enable administrators and users to create a one-to-one mapping between Windows users and their corresponding non-Windows credentials. When individual mappings are used, users can manage their own mappings. The SSO system maintains the one-to-one relation for the user's Windows account and the user's non-Windows account.

Windows end users can create and manage their own mappings for individual type applications. The same affiliate application can act as a Windows-initiated SSO and a Host-initiated SSO type application.

Important
Mappings can be created only for Windows domain accounts. Local accounts cannot be mapped.

Note
Only individual users can obtain the credentials to their individual accounts when individual mappings are used.

Group Mapping

SSO group mapping consists of mapping a Windows group, which contains multiple Windows users, to a single account in the affiliate application.

You can also specify multiple accounts for the SSO Application Users role. Each account that you specify can be associated with an external account.

For example, it is possible to map a domain user (domain\userA) to one set of external credentials (ExternalUser1). The same domain\userA could also be a member of Domain Group (domain\group1) and this group could be mapped to a different set of external credentials (ExternalUser2). In this case, it is important that the administrator (who must be an Application Administrator or above) specifies the correct order for the Application Users accounts.

The mapping for the first account (in the Order) of which the caller is a member is the one that will be used. In this case, if mapping for domain\userA to ExternalUser1 is set to Order 0, SSO will return this set of credentials for domain\UserA.

Only an application administrator, SSO affiliate administrator, or SSO administrator can create a group mapping.

You cannot specify the same group application for Windows-initiated SSO and Host-initiated SSO.

Important
Mappings can be created only for Windows domain accounts. Local accounts cannot be mapped.

Important
When you use group mappings, the members of the group can obtain the credential information for the group mapping.

See Also

Concepts

SSO Affiliate Applications

Understanding Enterprise Single Sign-On

Other Resources

Managing User Mappings

SSO Tickets

In an enterprise environment, where a user interacts with various systems and applications, it is very likely that the environment does not maintain the user context through multiple processes, products, and computers. This user context is crucial to providing single sign-on capabilities because it is necessary to verify who initiated the original request. To overcome this problem, Enterprise Single Sign-On (SSO) provides an SSO ticket (not a Kerberos ticket) that applications can use to obtain the credentials that correspond to the user who made the original request. By default, SSO tickets are not enabled. For more information about enabling tickets, see How to Configure the Enterprise Single Sign-On Tickets.

The SSO system issues a ticket when requested by an authenticated Windows user. The SSO system can only issue a ticket for the user making the request (you cannot request a ticket for other users). A ticket contains the encrypted domain and user name of the current user, and the ticket expiration time. After the SSO system issues a ticket, the ticket expires in two minutes by default. SSO administrators can modify the expiration time for tickets. For more information, How to Configure the Enterprise Single Sign-On Tickets.

After an application verifies the identity of the original requester, the application redeems the ticket to obtain the credentials of the user who initiated the request to the affiliate application. An application can redeem tickets from the SSO system in one of three ways:

- **Redeem only.** When an application initiates a request to redeem a ticket, the request must contain the name of the affiliate application to connect to, and the ticket itself. Only application administrators for the specific affiliate application, SSO affiliate administrators, or SSO administrators can redeem a ticket. You should use **Redeem only** when there is a trusted sub-system between the application that issued the ticket and the application that is redeeming the ticket. Only an application administrator for the specified affiliate application can redeem the ticket for a user.

- **Validate and redeem.** Tickets contain information about the user for whom the SSO system is performing the credential look-up. In this case, the SSO service verifies that the sender of the original message and the user of the ticket are the same before the system redeems the ticket.

An SSO administrator can disable ticket time-outs on a per-affiliate application basis. However, this is not recommended, as the ticket would never expire for this application. In scenarios that require that you disable ticket time-outs, ensure that there is a secure end-to-end trusted sub-system maintained between the front-end where the SSO system issues the ticket to the adapter where the SSO ticket redeems the ticket.

An SSO affiliate administrator can specify that tickets are allowed and that validation of the ticket is required on a per affiliate application basis. However, if the SSO administrator specifies at the SSO system level that the validation of tickets is required, the SSO affiliate administrator cannot turn off this option at the affiliate application level.

Important
When using SSO ticketing, you must ensure that the ticket time-out value is long enough to last between the time when the t icket is issued to the time that it is redeemed.

See Also

Tasks

How to Configure the Enterprise Single Sign-On Tickets

Concepts

Understanding Enterprise Single Sign-On

Other Resources

Managing User Mappings

Configuring Enterprise Single Sign-On

You can configure Enterprise Single Sign-On (SSO) using command-line utilities, user interface (UI) tools, or COM or Microsoft .NET Framework interfaces.

SSO Command-line Utilities

You use three different command-line utilities to perform Enterprise Single Sign-On tasks:

SSOConfig. Enables an SSO administrator to configure the Credential database and to manage the master secret.

Note
The Configuration Wizard creates the Credential database and the master secret server.

SSOManage. Enables SSO administrators, SSO affiliate administrators, and application administrators to update the Credential database to add, delete, and manage applications, to administer user mappings, and to set credentials for the affiliate application users. Some operations can be performed only by the SSO administrators, or, only by the SSO administrators and SSO affiliate administrators.

SSOClient. Enables Single Sign-On users to manage their own user mappings and set their credentials.

For more information about the SSO accounts, see Enterprise Single Sign-On User Groups.

SSO UI Tools

Enterprise SSO MMC Snap-in. Enables SSO administrators, SSO affiliate administrators, and application administrators to update the SSO database, to add, delete, and manage applications, to administer user mappings, and to set credentials for the affiliate application users. Some operations can be performed only by the SSO administrators, or only by the SSO administrators and SSO affiliate administrators. All operations that can be performed by the application administrators can also be performed by the SSO administrators and SSO affiliate administrators.

SSO Client Utility. Enables end users to manage their own mappings and set their credentials using the UI tool.

SSO COM and .NET interfaces

Enterprise Single Sign-On provides COM and Microsoft .NET Framework programmatic interfaces that enable you to create custom components, and to create scripts to facilitate the administration of the SSO system.

See Also

Concepts

SSO Components

Understanding Enterprise Single Sign-On

Other Resources

Using Enterprise Single Sign-On

Installing Enterprise Single Sign-On

The following sections contain information about the installation of the Enterprise Single Sign-On feature. Because of this feature's complex relationships to other features and systems, and because of its importance to system security, you should read this section carefully before you install Enterprise Single Sign-On.

It is also recommended that you review the latest software prerequisites for installing Enterprise Single Sign-On.

In This Section

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Older versions of Host Integration Server, such as Host Integration Server 2000 and SNA Server 4.0, provided both Single Sign-On and password synchronization through their host security feature. This feature was based around the host security domain, which contained user mappings to map credentials between Windows and host systems.

In Host Integration Server, the Enterprise Single Sign-On (SSO) feature replaces host security as the source of Single Sign-On and password synchronization. Although some concepts are shared between the old and new features, there are important differences. In addition to the increased functionality, there are two primary conceptual differences in the new features:

- Host security domains are replaced by affiliate applications.
- Security credential data is now stored in a SQL Server database.

To migrate your existing host security data into the new SSO environment, use the Migration Utility. This is a command-line tool (hissomig.exe) that migrates all necessary data from the old version to the new, enabling you to continue using Single Sign-On without modifying your applications.

The topics in this section walk you through the migration process. It is important to follow these steps in the order given.

In This Section

Back up the Existing Security Data

For safety purposes, you should back up your existing security data before you migrate to Enterprise Single Sign-On.

For information about how to do this, see the documentation for your particular product version (for example, Host Integration Server 2000 or SNA Server 4.0).

See Also

Concepts

SSO Security Recommendations

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Export the Encryption Key

If you are migrating from Host Integration Server 2000, you must export the encryption key. The encryption key is a security device that Host Integration Server 2000 uses to encrypt user passwords. It must be backed up because the migration utility will need it as part of the migration process.

To export the encryption key

1. Click **Start**, click **Run**, and then type **cmd**.
2. Locate the Host Integration Server 2000 installation directory.
3. Type **udbkey /showkey >hostseckey.bak** to back up the key into a file.
4. Protect the file securely.

See Also

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Install Enterprise Single Sign-On

After you back up the security data and export the encryption key in preparation for upgrading, it is time to install Enterprise Single Sign-On. You have several installation options. For more information, see Installing Enterprise Single Sign-On.

See Also

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Copy the Migration Utility to the Master Secret Server

The Migration Utility is not installed with the rest of Host Integration Server. You must manually copy it to the master secret server.

To copy the Migration Utility

1. Click **Start**, click **Run**, and then type **cmd** and press Enter.
2. Type **copy hisssomig.exe from <cddrive>\support\utilities\ to c:\hostsec** and press Enter.

See Also

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Run the Migration Utility

You can migrate your existing host security data into the new Single Sign-On (SSO) environment by using the hissomig.exe command-line utility. Migration is essentially a two-step process:

1. First, the tool **exports** data from the host security domain into an XML file. This file also contains validation data for the migration process. If mappings or file names conflict, an administrator can resolve them before the next step.

2. Second, the tool **imports** data into the Single Sign-On (SSO) environment, and updates the Host Integration Server credential database appropriately.

The Migration Utility has the following restrictions:

- You must be an SSO administrator and have at least read privileges to the SNAUDB database to perform migration.

- Both importing and exporting must be done on the master secret server.

Note
The XML file that is generated during migration is not deleted. Because this file contains security data, it is important that you manually delete the file as soon as migration is finished.

To export from Host Integration Server

1. Stop the Host Account Cache service.
2. Click **Start**, click **Run**, and then type **cmd** and press Enter.
3. Type **hissomig –export –servername <server> -output <XML file>** where <*server*> is the fully qualified name, NetBIOS name, or IP address of the Host Integration Server primary domain controller, and <*XML file*> is the full path of the XML file to which the data will be exported.

   For example:

   hissomig –export –servername SERVER1 -output c:\hostsecdb.xml
4. Press Enter.
5. Restart the Host Account Cache service.

To export from SNA Server 4.0

1. Stop the Host Account Cache service.
2. Click **Start**, click **Run**, and then type **cmd** and press Enter.
3. Type **hisssomig –export –dbfile <database file> -output <XML file>** where <*database file*> is the full path of the SNA 4.0 Host Security database file, and <*XML file*> is the full path of the XML file to which the data will be exported.

   For example:

   hisssomig –export –dbfile Y:\Program Files\SNA Server\hostsec\dbfile.dbs -output c:\hostsecdb.xml
4. Press Enter.
5. Restart the Host Account Cache service.

To import into Host Integration Server

1. Click **Start**, click **Run**, and then type **cmd** and press Enter.
2. Type **hisssomig –import –key <key> -input <XML file>** where <*key*> is the full path of the file that contains the encryption key, and <*XML file*> is the full path of the XML file from which the data will be imported.

   For example:

hisssomig –import –key Z:\hostseckey.bak -input c:\hostsecdb.xml

3. Press Enter.

Other commands for Migration Utility

The following is a list of commands for the Migration Utility. These commands are also displayed during migration if you attempt to run the utility with incorrect data.

Command	Comment
-servername	Name of server that holds the Host Integration Server Host Security primary host account cache database.
-dbfile	Full path of the SNA 4.0 Host Security database file.
-key	Full path of the file that contains the encryption key.
-username	User name that is used for encryption.
-password	Password for the user name that is used for encryption.
-output	Full path of the XML file to which the data will be exported.
-input	Full path of the XML file from which the data will be imported.
-log	Creates migration log in the directory specified. For example: **-log c:SSO\log**

See Also

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Upgrading from an Earlier Version of SSO

If you are installing the Enterprise Single Sign-on (SSO) feature, and you already have an earlier version deployed on your computer (for example, from Host Integration Server CTP, or Microsoft BizTalk Server), you must follow these steps.

- Back up the SSODB to a secure location.
- Back up the master secret key on the master secret server.
- Update the master secret server by running Host Integration Server Setup, selecting **Custom Installation**, and then selecting **Enterprise Single Sign-On**. After selecting **Enable Enterprise Single Sign-On on this computer**, select **Join an existing SSO system**.

You do not have to update the other SSO servers (non-master secret servers) from your BizTalk Server installation. However, if you want the new Enterprise Single Sign-On features to be available on those servers, you must update them by using the same procedures outlined earlier.

If you have a Beta version of Enterprise SSO installed as part of a Host Integration Server CTP release, you can use the same mechanism to upgrade those servers.

Note
These considerations also apply if you are installing Microsoft BizTalk Server 2006 on a computer that has an existing installation of Host Integration Server Enterprise Single Sign-On, and you want to update the servers.

See Also
Concepts
Using Host-Initiated SSO functionality in Enterprise Single Sign-On
Processing Servers for Enterprise Single Sign-On
Other Resources
Upgrading from Host Integration Server 2000 or SNA Server 4.0

Using Host-Initiated SSO functionality in Enterprise Single Sign-On

Host Initiated Single Sign-On (SSO) uses the protocol transition feature of Windows Server 2003 to perform Single Sign-On for the non-Windows user. This feature requires Windows Server 2003 and must be in a domain that has its **Domain Functional Level** set to **Windows Server 2003**.

See Also

Concepts

Upgrading from an Earlier Version of SSO

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Processing Servers for Enterprise Single Sign-On

In a multicomputer environment, after the master secret server and Single Sign-On (SSO) database have been created, you can install Enterprise Single Sign-On on subsequent computers. These are typically the computers on which either BizTalk Server or Host Integration Server is installed as well.

The initial installation process is the same as on the first computer. Configuration, however, becomes slightly different. Because the master secret server and the SSO database are already in place, select **Join** when the Configuration Wizard asks the question, **Create a new SSO system or Join an existing system?**

Note
During configuration, it is possible for a group on one computer to join an SSO database on a different computer that is not i n the database configured for that group. Although this is possible, it is not recommended.

See Also

Other Resources

Upgrading from Host Integration Server 2000 or SNA Server 4.0

Standard Installation Options

Host Integration Server takes advantage of the Enterprise Single Sign-On (SSO) capabilities for securely storing critical information.

By default, Enterprise Single Sign-On is not installed with the rest of Host Integration Server. You can install it by using the following procedure.

Note
When Enterprise Single Sign-on (Server component) is installed, configuration must be performed. The first step to set up an SSO system is to configure the master secret server. It is recommended that you set up a stand-alone master secret server. You can do this by only selecting Enterprise Single Sign-On from the custom feature tree in Host Integration Server Setup. It is also recommended that any computer that is running Enterprise Single Sign-On have a time synchronization service running. This keeps the computer time in sync with the rest of the system. This is necessary for SSO ticketing services to function correctly.

To install Enterprise Single Sign-On

1. Insert your Host Integration Server CD and run the Setup program again.
2. Select **Custom Installation**, and then select the appropriate option from the following list:

   When you run the HIS Server package, you have the following options:

   - **Enterprise Single Sign-On Master Secret Server —** Acts as the master secret server in the SSO system. This is the first server in the SSO system that must be deployed. This enables you to create the SSO Credential Database.
   - **Enterprise Single Sign-On Administration —** Administration and client tools for mapping and connecting to Enterprise Single Sign-On services.
   - **Server Runtime —** Core services to enable single sign-on and to store/access configuration data securely.
   - **Enterprise Single Sign-On Services with Password Synchronization —** Services to enable the Password Synchronization feature in the Enterprise SSO system. These services also integrate with the Microsoft Password Change Notification Service. Once you have installed the core Enterprise Single Sign-On services, you can install the Password Synchronization feature of Enterprise SSO from the Host Integration Server package by starting Setup (\Platform\SSO\Setup.exe) and selecting the **Password Synchronization** feature.
   - **Enterprise Single Sign-On Services —** Provides the core services to enable single sign-on and to store/access configuration data securely. Can act as the master secret server in the SSO system.
   - **Enterprise Single Sign-On Services with Password Synchronization —** Provides the services to enable the Password Synchronization feature in the Enterprise SSO System. These services also integrate with the Microsoft Password Change Notification Service.
   - **Enterprise Single Sign-On Administration —** Administration and client tools for mapping and connecting to Enterprise Single Sign-On services.

   When you run the HIS Client package, you have the following options:

   - **Enterprise Single Sign-On Administration —** Administration and client tools for managing and connecting to Enterprise Single Sign-On services.
   - **Enterprise Single Sign-On Client —** Client tools for end users to manage their mappings.

See Also
Tasks

How to Install the Enterprise Single Sign-On Administration Component
How to Install the Enterprise Single Sign-On Client Utility
How to Remove Enterprise Single Sign-On

Other Resources

High-Availability Installation Options

How to Install the Enterprise Single Sign-On Administration Component

You can install the Enterprise Single Sign-On (SSO) Administration component as a stand-alone feature. This is useful if you need to administer the SSO system remotely. The hardware and software requirements are the same as for a typical Enterprise SSO installation.

After you install the administration component, you must use either ssomanage.exe or the SSO Administration MMC Snap-In to specify the SSO server that will be used for management. Both processes are included in the procedure that follows.

Note
While the SSO Administration feature in Host Integration Server is compatible with the server version of SSO in BizTalk Server 2006, the administrative components of Enterprise SSO in BizTalk Server 2006 are not compatible with the server version of Enterprise SSO in Host Integration Server.

Installing the SSO administrative utility (ssomanage.exe) does not create shortcuts on the **Start** menu that let you access the command-line utilities. To run the SSO administrative utilities after installation, you must open a command prompt and navigate to the SSO directory located at Program Files\Common Files\Enterprise Single Sign-On.

The Enterprise SSO Administration feature also includes an MMC Snap-in. The Snap-in is installed on Windows Server 2003 and Windows XP. It is not supported on Windows 2000. You must also have MMC 3.0 installed on your computer for the Snap-in to function.

To open the Enterprise SSO MMC Snap-in, click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.

To install the Enterprise Single Sign-On administrative component

1. Perform a custom installation of Host Integration Server, selecting only the Enterprise Single Sign-On administration component.
2. When the installation program finishes, click **Start**, click **Run**, and then type **cmd**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Do one of the following:

   - Type **ssomanage –server** to specify the SSO server that you want to connect to when you perform administration operations.

   OR

   Type **ssomanage -serverall** to specify the SSO server that all users of this computer will connect to when performing administration operations.

   OR

   Open the ENTSSO Administration MMC Snap-In. The Select SSO Server dialog will appear. Enter or browse to the SSO Server desired. To specify the SSO Server for all users on the machine, select **Set SSO Server for all users**.

See Also

Tasks

How to Install the Enterprise Single Sign-On Client Utility

Standard Installation Options

Concepts

Configuring Enterprise Single Sign-On

Other Resources

Installing Enterprise Single Sign-On

How to Install the Enterprise Single Sign-On Client Utility

The stand-alone Single Sign-On (SSO) client utility (ssoclient.exe) enables end users to configure their client mappings in the credential database. You can install the client utility from a self-extracting file (SSOClientInstall.exe), which is installed with the SSO administration feature. Administrators can also make the installer package available to client users by placing a copy of the installer package on a network share.

To install the SSO client utility, you must be running one of the following operating systems on the client computer:

- Windows Server 2003
- Windows 2000 Server with Service Pack 4, or Windows XP Professional with Service Pack 1
- .NET Framework 2.0 (only necessary for the managed interoperability component of the SSO client utility)

Installing the SSO client utility does not create shortcuts on the **Start** menu for you to access the command-line utilities. To run the SSO client utility after installation, you must open a command prompt and navigate to the SSO directory located at Program Files\Common Files\Enterprise Single Sign-On.

To install the SSO client utility

1. Double-click the installer package, SSOClientInstall.

Note
Ask your Enterprise Single Sign-On administrator for the location of this file in your enterprise.

   The WinZip Self-Extractor program appears.
2. Select the folder where you want to unzip the files, and then click **Unzip**.

   It is recommended that you unzip the files in a temporary folder.

   The **Enterprise Single Sign-On Client Setup** program appears.
3. On the **Welcome to the Enterprise Single Sign-On Client** page, click **Next**.
4. On the **License Agreement** page, click **I accept the terms of this license agreement**, and then click **Next**.
5. On the **User Information** page, type your user name, organization name, and then click **Next**.
6. On the **Start Installation** page, click **Install**.
7. On the **Completing the Enterprise Single Sign-On Client Wizard** page, click **Finish**.
8. Specify the SSO server by doing either of the following:

   - Type **ssomanage –server** to specify the SSO server that you want to connect to when you perform administration operations. You can also type **ssomanage -serverall** to specify the SSO server that all users of this computer will connect to when performing administration operations.

     OR

   - Open the ENTSSO Administration MMC Snap-In. The Select SSO Server dialog will appear. Enter or browse to the SSO Server desired. To specify the SSO Server for all users on the machine, select **Set SSO Server for all users**.

See Also
Tasks
How to Install the Enterprise Single Sign-On Administration Component
Concepts
Configuring Enterprise Single Sign-On
Other Resources
Installing Enterprise Single Sign-On

High-Availability Installation Options

Topics in this section describe installation focused on high availability of Enterprise Single Sign-On, such as multicomputer deployment.

In This Section

How to Cluster the Master Secret Server

How to Cluster the SQL Server

How to Configure Enterprise Single Sign-On in a Multicomputer Scenario

How to Cluster the Master Secret Server

It is strongly recommended that you follow the instructions in this section to cluster the Enterprise Single Sign-On (SSO) service on the master secret server successfully.

Before you start configuring SSO in a cluster environment, it is recommended that you understand how clustering works. For more information, see http://go.microsoft.com/fwlink/?LinkId=33180.

When you cluster the master secret server, the Single Sign-On Servers communicate to the active clustered instance of the master secret server. Similarly, the active clustered instance communicates with the Credential database.

You must be an SSO administrator to perform this procedure.

Caution
You cannot install the master secret server on a Network Load Balancing (NLB) cluster.

To cluster the master secret server

1. Perform a custom installation to install the master secret server on the first node of the cluster (for example, ClusterNode1).
2. In the **Configuration Wizard**, on the **Configuration Questions** page, in the **Is this the master secret server?** list, select **Yes**, and then click **Next**.
3. Specify the service account credentials for SSO service. This must be a member of the SSO Administrators group account.
4. Specify the location of the SQL Server and SSO Credential database (SSODB).
5. Back up the master secret on the active node.
6. Perform a custom installation to install the master secret server on the second node of the cluster (ClusterNode2).
7. Use the **Configuration Wizard** to configure an Enterprise SSO Server on the second node of the cluster. This time, however, select **No** when you reach the question in step 2, because this is not the initial installation of the master secret server.
8. Click **Next**, and then complete the **Configuration Wizard**.
9. Stop the SSO service by typing **net stop entsso** at the command line.
10. Change the master secret server name in the SSO Credential database to the cluster name. For example, if the cluster is named MSS_CLUSTER, change the name from ClusterNode1 to MSS_CLUSTER.
11. Use a text editor to copy and paste the following code into an .xml file (for example: MSS CLUSTER.xml) and save the file:

    ```
    <sso>
      <globalInfo>
         <secretServer>MSS_CLUSTER</secretServer>
    </globalInfo>
       </sso>
    ```

12. At the command line, navigate to the Enterprise Single Sign-On installation directory. The default is Program Files\Common Files\Enterprise Single Sign-On.
13. Type **ssomanage -updatedb <filename>** where *filename* is the name of the .xml file in the previous step. This updates the master secret server name in the database.

    Ignore any run-time errors. The Microsoft Distributed Transaction Coordinator (DTC) was not configured to run on a cluster, and may be unable to start.
14. Open a command prompt on master secret server 1 and type **comclust -a**.
15. From the **Services** console, right-click **Distributed Transaction Coordinator**, and then click **Restart**.

16. Open a command line on master secret server 2 and type **comclust -a**.
17. From the **Services** console, right-click **Distributed Transaction Coordinator**, and then click **Restart**.
18. Open **Cluster Administrator**, and then click the cluster group that has the master secret server cluster.
19. On the **File** menu, point to **New**, and then click **Resource**.

    The New Resource window opens.

    Under **Name**, type the name of the SSO resource (for example, **ENTSSO**).

    Under **Resource Type**, select **Generic Service**.
20. In the **Possible Owners** window, include each cluster node as a possible owner of the ENTSSO resource.
21. After you create the **ENTSSO** resource, right-click **ENTSSO**, and click **Properties**.
22. In the **Cluster Properties** dialog box, click the **Security** tab, and verify that the user under which the application is running has sufficient user rights (not a local administrator) to access the cluster.
23. Open **Cluster Administrator**, right-click the cluster group that has the master secret server cluster, and then click **Move group**.

    This moves the master secret server resources from the first node to the second node.
24. Click **Start**, click **Run**, and then type **cmd**.
25. At the command prompt, navigate to the Enterprise Single Sign-On installation directory. The default is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
26. Type **ssoconfig -restoresecret <restore file>**, where <*restore file*> is the path and name of the back-up file that contains the master secret.

See Also
Concepts
Master Secret Server
Other Resources
Installing Enterprise Single Sign-On
High-Availability Installation Options

How to Cluster the SQL Server

For information about how to cluster the SQL server, see your SQL Server documentation.

See Also

Other Resources

High-Availability Installation Options

How to Configure Enterprise Single Sign-On in a Multicomputer Scenario

This section contains instructions for configuring Enterprise Single Sign-On (SSO) in a three-computer scenario.

In the following scenario, computer A is the master secret server, computer B is the Single Sign-On server, and computer C holds the Credential database. Computer B can act as a runtime server, as an administration server (administration sub services of SSO use this server for managing the Credential database), or as a mapping server (administration and client sub services of SSO use this server to manage mappings).

If you want to add more SSO servers to your environment, follow the steps for configuring computer B. Any new SSO servers will point to the existing Credential database, and cannot be the master secret server.

Note
It is recommended that you configure the master secret server on a different computer from the Single Sign-On server and t he Credential database.

To configure the master secret server and create the Credential database on Computer A

1. Perform a custom installation of Host Integration Server, and install only the Enterprise Single Sign-On runtime component.
2. Run the Configuration Wizard to configure SSO on the master secret server.

   On the **Configuration Questions** page, in the **Is this the master secret server** list, select **Yes**, and then click **Next**.
3. On the **Windows Accounts** page, specify the service account credentials for the SSO service. This must be a member of the SSO Administrators group.
4. On the **Database Configurations** page, specify the location of the SQL server (computer C) and the name of the Credential database (SSODB).
5. Back up the master secret.

   For more information, see How to Back Up the Master Secret.

To configure the SSO server on Computer B

1. Install Enterprise Single Sign-On on Computer B.

Note
This can be a Host Integration Server computer, or an SSO-only server (SSO runtime components).

2. Run the Configuration Wizard to configure SSO.

   On the **Configuration Questions** page, in the **Is this the master secret server** list, select **No**, and then click **Next**.
3. On the **Windows Accounts** page, specify the service account credentials for the SSO service. This must be a member of the SSO Administrators group.
4. On the **Database Configurations** page, point to the location of the SQL server (computer C) and the name of the Credential database (SSODB).

See Also

Tasks

How to Cluster the Master Secret Server

Other Resources

Installing Enterprise Single Sign-On

High-Availability Installation Options

How to Remove Enterprise Single Sign-On

If you remove Host Integration Server, Enterprise Single Sign-On (SSO) is no longer configured, but it is not removed. You must remove SSO separately. You can also restore configuration information including the master secret to reuse existing data. For more information, see How to Restore the Master Secret.

To remove Enterprise Single Sign-On

1. Back up the master secret key.

   For more information, see How to Back Up the Master Secret.

2. Uninstall Host Integration Server.
3. Click **Start**, point to **Settings**, and then click **Control Panel**.
4. Click **Add or Remove Programs**.
5. In **Add or Remove Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **Remove**.
6. Click **Yes** when you are prompted to confirm the removal of Microsoft Enterprise Single Sign-On.

See Also

Other Resources

Installing Enterprise Single Sign-On

Using Enterprise Single Sign-On

You can use either the Microsoft Management Console (MMC) Snap-in or the command-line management utility (ssomanage) to manage the Single Sign-On (SSO) system. This includes activities such as updating the SSO database, adding, deleting, and managing applications, and administering user mappings.

Only Single Sign-On Administrators can perform these tasks.

In This Section

- How to Set the Enterprise Single Sign-On Server
- How to Enable Enterprise Single Sign-On
- How to Change the Master Secret Server
- How to Disable Enterprise Single Sign-On
- How to Update the Credential Database
- How to Display the Credential Database Information
- How to Configure the Enterprise Single Sign-On Tickets
- How to Audit Enterprise Single Sign-On
- How to Enable SSL for Enterprise Single Sign-On
- Managing the Master Secret
- Specifying Single Sign-On Administrators and Affiliate Administrators Accounts
- Managing Affiliate Applications
- Managing User Mappings
- Host Initiated Single Sign-On
- How to Use the ENTSSO Management Agent
- How to Use the Servers Snap-In
- How to Use Direct Password Sync
- How to Use the Mapping Wizard
- How to Use Password Filters

How to Configure SSO Using the Configuration Wizard

Use the **Enterprise Single Sign-On (SSO)** page to configure your SSO settings.

Note
You will not be able to reconfigure SSO in the configuration manager after you have configured it.

Use this	To do this
Enable Enterprise Single Sign-On on this computer	Select **Enable Enterprise Single Sign-On on this computer** to configure this server with SSO settings.
Create a new SSO system	Select **Create a new SSO system** if this is the first SSO server you are configuring in your SSO system. This also creates and configures the SSO Credential database. You must also back up the secret on this secret server. Caution You should configure the master secret server as a stand-alone server. You must be an SSO administrator while performing this configuration task. Note Only one master secret server can be associated with one SSO group. Associating two master secret servers to the same SSO group is not supported.
Join an existing SSO system	Select **Join an existing SSO system** to connect to an existing SSO system.
Data stores	The **Data stores** list provides an editable view of the data stores used for the SSO database.
Windows service	The **Windows service** list provides an editable view of the account used to run the Enterprise Single Sign-On service.
Windows accounts	The **Windows accounts** list provides an editable view of the SSO Administrators and SSO Affiliate Administrators Windows groups.

Use the **Enterprise Single Sign-On Secret Backup** page to save the master secret to an encrypted backup file in the event that disaster recovery is needed.

Use this	To do this
Secret backup password	Type the password for the backup file.
Confirm password	Confirm the password for the backup file.
Password reminder	Type a reminder for the password you enter.
Backup file location	Provide a file name and file path to the secret backup file. By default it is stored at <drive>:\Program Files\Common Files\Enterprise Single Sign-On\.

Considerations for Configuring SSO

When you configure Enterprise SSO, consider the following:

- When configuring the SSO Windows accounts using local accounts, you must specify the account name without the

computer name.

- When using a local SQL Server named instance as data store, you must use LocalMachineName\InstanceName instead of LocalMachineName\InstanceName, PortNumber.

- It is possible to configure the Enterprise SSO service account as the Network Service account. To do so, you must first add the Network Service account to the SSO Administrators group and reboot the computer for the change to take effect.

Important
While it is possible to do this, it is not a good security practice, as the Network Service account is a low privilege account.

How to Set the Enterprise Single Sign-On Server

Every time that you use the command line management utility, ssomanage, you must first point the user to the Single Sign-On (SSO) server that you want to connect to.

You can do this in one of two ways:

- Individual users can point themselves to the correct Single Sign-On server.
- A local computer administrator for the Single Sign-On server can point all the members of the Single Sign-On Users account to this server.

To set the Enterprise Single Sign-On server using the MMC Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the MMC Snap-In under the **Console Root**, right-click **Enterprise Single Sign-On**, and then click **Select**.
3. Browse to the desired server.
4. If appropriate, select the **Set SSO Server for all users** check box.
5. Click **OK**.

To set the Enterprise Single Sign-On server for a single user

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –server <Single Sign-On Server>**, where <*Single Sign-On Server*> is the computer name of the Single Sign-On server that the user wants to connect to.

To set the Enterprise Single Sign-On server for all users

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –serverall <Single Sign-On Server>**, where <*Single Sign-On Server*> is the computer name of the Single Sign-On server that all members of the Single Sign-On Users account will be pointed to.

To determine the Enterprise Single Sign-On Server to which a user is connected

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –showserver**.

Note
This command displays the settings for the current user and also for other users if they exist.

See Also
Tasks
How to Enable Enterprise Single Sign-On
How to Disable Enterprise Single Sign-On
How to Display the Credential Database Information
Other Resources
Using Enterprise Single Sign-On

How to Enable Enterprise Single Sign-On

The enabling command enables the entire Enterprise Single Sign-On (SSO) system. After you run the enabling command, there is a short delay before all Single Sign-On servers are enabled, because each server polls the Credential database for the latest global information.

If you want to configure affiliate applications and mappings in the SSO system, you must also create an affiliate application. After an SSO affiliate administrator creates an affiliate application, an application administrator can make changes to it, and application users (end users) can create their own mappings. For more information, see Managing Affiliate Applications and Managing User Mappings.

To enable the SSO system using the MMC Snap-In

1. Click **Start**, click **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Enable**.

To enable the SSO system using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –enablesso**.

To enable SSO to create affiliate applications and mappings

1. Log on as an SSO administrator or SSO affiliate administrator to the SSO Server, or on a computer that has the SSO administration sub services of SSO.
2. Click **Start**, click **Run**, and then type **cmd**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Type **ssomanage -enablesso** to enable the Enterprise Single Sign-On service.
5. Log on as an SSO affiliate administrator.
6. Type **ssomanage -createapps <application file>** to create an affiliate application, where <*application file*> is the XML file that contains definitions for the affiliate applications.

See Also

Tasks

How to Set the Enterprise Single Sign-On Server
How to Disable Enterprise Single Sign-On
How to Update the Credential Database

Other Resources

Using Enterprise Single Sign-On

How to Change the Master Secret Server

After you set up the master secret server and configure the Credential database, you can change the master secret server if the original master secret server fails and cannot be recovered. To change the master secret server, you must promote a Single Sign-On (SSO) server to become the master secret server.

To change the master secret server using the MMC Snap-in

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane, right-click **System**, and then click **Properties**.

   The master secret server is displayed on the **General** tab of the **SSO System Properties** dialog box.
3. Click **Change** to select a new master secret server.

To promote a Single Sign-On server to master secret server

1. Create an XML file that includes the name of the SSO server that you want to promote to master secret server. For example:

```
<sso>
  <globalInfo>
     <secretServer>SSO Server name</secretServer>
  </globalInfo>
</sso>
```

2. Click **Start**, click **Run**, and then type **cmd**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Type **ssomanage –updatedb <update file>**, where <*update file*> is the name of the XML file that you create in step 1.
5. Restart the master secret server.
6. Type **ssoconfig –restoresecret <restore file>**, where <*restore file*> is the path and name of the file where the master secret is stored.

See Also

Tasks

How to Cluster the Master Secret Server

How to Update the Credential Database

Concepts

Master Secret Server

Other Resources

Using Enterprise Single Sign-On

How to Disable Enterprise Single Sign-On

The disabling command disables the entire Single Sign-On system.

There will be a short delay before all Single Sign-On servers are disabled, because they poll the Credential database for the latest global information.

To disable Enterprise Single Sign-On using the MMC Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Disable**.

To disable Enterprise Single Sign-On using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –disablesso**.

See Also

Tasks

How to Enable Enterprise Single Sign-On

Other Resources

Using Enterprise Single Sign-On

How to Update the Credential Database

You use the commands described here to change the global information in the Credential database, such as the master secret server identification, the account names, auditing in the database, ticket time-out, credential cache time-out, and so on.

Changing Time-outs for the SSO System

You can modify two time-outs at the Enterprise Single Sign-On (SSO) system level:

Ticket timeout. This property specifies the length of time for which a ticket that SSO issues is valid. To satisfy most of the scenarios in an enterprise that use SSO, the default ticket time-out is two minutes. The SSO administrator can change this based on the application requirements.

Credential Cache timeout. This property specifies the credential cache time-out for all SSO servers. SSO servers cache the credentials after the first lookup. By default, the credential cache time-out is 60 minutes. The SSO administrator can change this to an appropriate value based on the security requirements.

You change both of these time-outs by updating the Credential database.

The following is an example XML file for updating the Credential database:

```
<sso>
<globalnfo>
<ssoAdminAccount>Domain\Accountname</ssoAdminAccount>
<ssoAffiliateAdminAccount>Domain\Accountname</ssoAffiliateAdminAccount>
<secretServer>ComputerA</secretServer>
<auditDeletedApps>1000</auditDeletedApps>
<auditDeletedMappings>1000</auditDeletedMappings>
<auditCredentialLookups>1000</auditCredentialLookups>
<ticketTimeout>2</ticketTimeout>
<credCacheTimeout>60</credCacheTimeout>
</globalInfo>
</sso>
```

To change time-outs using the Microsoft Management Console (MMC) Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Properties**.
4. On the **SSO System Properties** dialog box, click the **General** tab.
5. Enter the appropriate settings, and then click **OK**.

To update the SSO database using the MMC Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Update**.

To update the Credential database using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –updatedb <update file>**, where <*update file*> is the path and name of the file.

See Also

Tasks

How to Configure the Enterprise Single Sign-On Tickets

Other Resources

Using Enterprise Single Sign-On

How to Display the Credential Database Information

You can view Single Sign-On (SSO) credential database information by using the ssomanage command.

To display the SSO database information using the Microsoft Management Console (MMC) Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Properties**.
4. Click the tabs in the **SSO System Properties** dialog box to view SSO database information.

To display the Credential database information using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –displaydb**.

To display the Credential database that the SSO server is connected to

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –showdb**.

The following table describes the values that are displayed.

Property	**Value**
SQL Server	*<SQL Server name>*
SQL Server database	*<SQL Server database name>*
Single Sign-On Secret Server name	*<Single Sign-On Server name>*
Single Sign-On Administrators account	Domain\account name
Single Sign-On Affiliate Administrators account	Domain\account name
Size of audit table for deleted applications (number of audit entries)	1000 (default)
Size of audit table for deleted user mappings (number of audit entries)	1000 (default)
Size of audit table for external credential lookups (number of audit entries)	1000 (default)
Ticket timeout (in minutes)	2 (default) This value can be an integer between 1 to 525600
Credential cache timeout (in minutes)	60 (default)
Single Sign-On Status	Enabled/disabled
Tickets allowed	Yes/No (default)

Validate tickets	Yes (default)/No

See Also

Tasks

How to Configure the Enterprise Single Sign-On Tickets

Other Resources

Using Enterprise Single Sign-On

How to Configure the Enterprise Single Sign-On Tickets

You can use the Microsoft Management Console (MMC) Snap-In or the command line to control ticket behavior for the entire Single Sign-On (SSO) system, including whether to allow tickets, and whether the system must validate the tickets. You can use **Yes**, **No**, **On**, or **Off** to indicate whether to allow or validate tickets. These words are case independent, and they must be used regardless of your language settings.

If you have the SSO Administration feature installed on a remote computer, you can perform remote IssueTicket operation. Note that all traffic between the SSO Administration module and the Runtime module (ENTSSO service) is encrypted.

By using the command-line utility, ssomanage.exe, you can specify the ticket time-out at the affiliate application level only when an update of the application is performed, not at creation time. Only an SSO Administrator can configure tickets at the SSO system level and at the affiliate application level. If ticketing is disabled at the system level, it cannot be used at the affiliate application level either. You can enable tickets at the system level and disable them at the affiliate application level. If validation is enabled at the system level, validation of tickets is required at the affiliate application level also. You can disable validation at the system level and enable it at the affiliate application level.

If ticket time-out is specified both at the system level and the affiliate application level, the one specified at the affiliate application level is used to determine the ticket expiry time.

For more information about tickets and ticket validation, see SSO Tickets.

To configure the SSO tickets using the MMC Snap-In for the Affiliate Application

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Affiliate Applications** node.
3. Right-click **Affiliate Application**, and then click **Properties**.
4. Click the **Options** tab.
5. Select **Allow Tickets** and configure the ticket time-out as appropriate.

To configure the SSO system-level tickets using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –tickets <allowed yes/no> <validate yes/no>**, where <*allowed yes/no*> indicates whether tickets will be allowed or not, and <*validate yes/no*> indicates whether tickets must be validated after they are redeemed.

Note
You can use **yes**, **no**, **on**, or **off** to indicate whether to allow or validate tickets. These words are case independent, and t hey must be used regardless of your language settings.

See Also
Concepts
Understanding Enterprise Single Sign-On
Other Resources
Using Enterprise Single Sign-On

How to Audit Enterprise Single Sign-On

Use this command to set both the positive and negative auditing levels. Single Sign-On (SSO) administrators can set the positive and negative audit levels that suit their corporate policies. You can set positive and negative audits to one of the following levels:

- 0 = None
- 1 = Low
- 2 = Medium
- 3 = High - This level issues as many audit messages as possible.

The default value for positive auditing is 0 (none), and the default value for negative auditing is 1(low).

To change the database-level auditing, you must update the Credential database using an XML file. The following is an example XML file that is used for updating the Credential database:

```
<sso>
<globalnfo>
<auditDeletedApps>1000</auditDeletedApps>
<auditDeletedMappings>1000</auditDeletedMappings>
<auditCredentialLookups>1000</auditCredentialLookups>
</globalInfo>
</sso>
```

To audit Single Sign-On using the Microsoft Management Console (MMC) Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Properties**.
4. In the **SSO System Properties** dialog box, click the **Audits** tab.
5. Enter the appropriate settings, and then click **OK**.

To audit Single Sign-On using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoconfig –auditlevel < positive level> <negative level>**, where <*positive level*> is the level of auditing when actions succeed, and <*negative auditing*> is the level of auditing when actions fail.

To audit the Credential database

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –updatedb <update file>**, where <*update file*> is the path and name of the file.

See Also

Tasks

How to Update the Credential Database

Other Resources

How to Enable SSL for Enterprise Single Sign-On

Use the following command to enable Secure Sockets Layer (SSL) between all the Enterprise Single Sign-On (SSO) servers and the Credential database.

To enable SSL for Enterprise Single Sign-On

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –setssl <yes/no>**, where <*yes/no*> indicates whether you want to enable SSL in the SSO system.

See Also
Other Resources
Using Enterprise Single Sign-On

Managing the Master Secret

The master secret is the key that is used to encrypt all the information stored in the Credential database. If the master secret server crashes and you lose the secret, you cannot retrieve the information stored in the Credential database. Therefore, it is very important to back up the master secret as soon as you generate it.

In This Section

- How to Generate the Master Secret
- How to Back Up the Master Secret
- How to Restore the Master Secret

How to Generate the Master Secret

You must have administrator rights on the master secret server in order to generate the master secret. In addition, you must perform this task from the master secret server.

The first server where you install Enterprise Single Sign-On (SSO) becomes the master secret server.

Important
There can be only one master secret server in your SSO system.

Note
When Enterprise Single Sign-On is installed as part of the Host Integration Server installation, the master secret is generated as part of the Configuration Wizard. You only need to perform this task if you want to generate a new master secret.

To generate the master secret using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO Microsoft Management Console (MMC) Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Generate Master Secret**.

To generate the master secret using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoconfig –generatesecret <backup file>**, where <*backup file*> is the name of the file that contains the master secret.

   You will be prompted to enter a password to protect the file you just created.

See Also
Tasks
How to Back Up the Master Secret
Concepts
Master Secret Server
Other Resources
Managing the Master Secret

How to Back Up the Master Secret

You can back up the master secret from the master secret server onto an NTFS file system or removable media, such as a floppy disk.

You must be a Single Sign-On administrator and a Windows administrator to perform this task. The Single Sign-On (SSO) system will prompt you for a password. To restore the secret later, you must specify the same password.

Caution
If the master secret server crashes and you lose the key, or if the key becomes corrupted, you will not be able to retrieve data stored in the Credential database. You must back up the master secret, or you risk losing data from the Credential database.

To back up the master secret using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO Microsoft Management Console (MMC) Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Back up Master Secret**.

To back up the master secret using the command line

1. On the **Start** menu, click **Programs**, and then click **Accessories**. Right-click **Command Prompt**, and then click **Run As...**.
2. Select the appropriate Administrator, and then click **OK**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Type **ssoconfig –backupsecret <backup file>**, where <*backup file*> is the path and name of the file where the master secret will be backed up, for example, **A:\ssobackup.bak**.
5. Provide a password to help protect this file.

   You will be prompted to confirm the password and to provide a password hint to help you remember this password.

Important
You must save and store the backup file in a secure location.

See Also

Tasks

How to Generate the Master Secret

How to Restore the Master Secret

Concepts

Master Secret Server

Other Resources

Managing the Master Secret

How to Restore the Master Secret

As part of data recovery procedures, you might have to restore the master secret to reuse existing data. To perform this task, you must log on to the master secret server by using an account that is both a Windows administrator and a Single Sign-On (SSO) administrator.

To restore the master secret using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO Microsoft Management Console (MMC) Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Restore Master Secret**.

To restore the master secret using the command line

1. On the **Start** menu, click **Programs**, and then click **Accessories**. Right-click **Command Prompt**, and then click **Run As...**.
2. Select the appropriate Administrator, and then click **OK**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Type **ssoconfig –restoresecret <restore file>**, where <*restore file*> is the path and name of the file where the master secret is stored.

See Also

Tasks

How to Generate the Master Secret

How to Back Up the Master Secret

Concepts

Master Secret Server

Other Resources

Managing the Master Secret

Specifying Single Sign-On Administrators and Affiliate Administrators Accounts

The Enterprise Single Sign-On (SSO) Administrators and Affiliate Administrators accounts can be host group or individual accounts. You must create these accounts before you configure the SSO system. When you are using domain accounts, you must create the SSO Administrators and SSO Affiliate Administrators accounts as domain global security groups in the domain controller. You must be a domain administrator to create these accounts.

You must specify the Single Sign-On Administrators and Affiliate Administrators accounts in the Credential database. The following example shows XML code that can be used to update the Credential database:

```
<sso>
<globalInfo>
<ssoAdminAccount>Domain\Accountname</ssoAdminAccount>
<ssoAffiliateAdminAccount>Domain\Accountname</ssoAffiliateAdminAccount>
</globalInfo>
</sso>
```

Note

The Configuration Wizard automatically specifies the SSO Administrator and SSO Affiliate Administrator groups in the Credential database.

Before you update the Credential database with the SSO Administrators group, you must disable the Single Sign-On system. You can use the Microsoft Management Console (MMC) Snap-In or the command line to do this.

To disable the Enterprise Single Sign-On system using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Disable**.

To disable the Enterprise Single Sign-On system using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory. The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –disablesso**.

To update the SSO database using the MMC Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Update**.

To update the Credential database using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –updatedb <update file>**, where <*update file*> is the path and name of the XML file.

To enable the Enterprise Single Sign-On system using the MMC Snap-In

1. Click **Start**, point to **Programs**, point to **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.

2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Enable**.

To enable the Enterprise Single Sign-On system using the command line

1. Click **Start**, click **Run**, and then type **cmd**.
2. At the command line, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –enablesso**.

See Also

Concepts

Enterprise Single Sign-On User Groups

Other Resources

Using Enterprise Single Sign-On

Managing Affiliate Applications

This section provides information on how to create and configure affiliate applications.

In This Section

- How to Create an Affiliate Application
- How to Delete an Affiliate Application
- How to Update the Properties of an Affiliate Application
- How to Enable an Affiliate Application
- How to Disable an Affiliate Application
- How to List Affiliate Applications
- How to List the Properties of an Affiliate Application
- How to Clear the Application Cache
- How to Set the Enterprise SSO Server Using the Client Utility
- How to Display the Enterprise SSO Server Using the Client Utility
- How to Set Credentials for the Affiliate Application Using the Client Utility

How to Create an Affiliate Application

You can use the MMC Snap-In or the **createapps** command to create one or more applications, as specified by the XML file. The following is an example XML file for Windows-Initiated Single Sign-On (SSO):

```
<sso>
<application name="SAP">
<description>The SAP application</description>
<contact>someone@example.com</contact>
<appuserAccount>domain\AppUserAccount</appuserAccount>
<appAdminAccount>domain\AppAdminAccount</appAdminAccount>
<field ordinal="0" label="User Id" masked="no" />
<field ordinal="1" label="Password" masked="yes" />
<flags groupApp="no" configStoreApp="no" allowTickets="no" validateTickets="yes" allowLocal
Accounts="no" timeoutTickets="yes" adminAccountSame="no" enableApp="no" />
</application>
</sso>
```

After you create an affiliate application, you cannot modify the following properties:

- Name of the affiliate application.
- Fields associated with the affiliate application.
- Affiliate application type (host group, individual, or configuration store).
- Administration account same as affiliate administrators group. (Specifying this flag will always use the affiliate administrators group as the administrator account for this affiliate application.)

Important

You can use local accounts for the administration account and user account by setting the allowLocalAccounts flag to yes. However, you should only use this flag in single-computer scenarios.

Security Note

You must be an SSO administrator or SSO affiliate administrator to perform this task.

Note

If you are performing the configuration on a Domain Controller, and the Domain Local scope groups are specified for Application Administrators or Application Users while creating Affiliate Applications, it is recommended that you enable the local account flag. To do this:

- In the MMC Snap-in, select Allow local accounts for access accounts during the creation process.
- From the command line, specify allowLocalAccounts=yes in the XML file for Affiliate Application creation.

After you create the affiliate application, you must enable it. For more information, see How to Enable an Affiliate Application.

To create an affiliate application using the Microsoft Management Console (MMC) Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **Affiliate Applications**, and then click **New**.

4. Follow the instructions in the **Create New Affiliate Application** wizard.

To create an affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createapps <application file name>**, where <*application file name*> is the XML file.

See Also

Tasks

How to Enable an Affiliate Application

How to Delete an Affiliate Application

Concepts

SSO Affiliate Applications

Other Resources

Managing User Mappings

Managing Affiliate Applications

How to Delete an Affiliate Application

Use the MMC Snap-In or the **deleteapps** command to delete the specified affiliate application from the Credential database.

Important
When you delete an affiliate application, the SSO system automatically deletes all mappings associated with it.

Security Note
You must be an SSO administrator or SSO affiliate administrator to perform this task.

To delete an affiliate application using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click the affiliate application, and then click **Delete**.

To delete an affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –deleteapp <application name>**, where <*application name*> is the name of the affiliate application you want to remove from the Credential database.

See Also
Tasks
How to Enable an Affiliate Application
Concepts
SSO Affiliate Applications
Other Resources
Managing User Mappings
Managing Affiliate Applications

How to Update the Properties of an Affiliate Application

Use the MMC Snap-In or the **updateapps** command to update one or more application properties, as specified by the XML file. You must be an affiliate administrator to perform this task. The following is an example XML file that lists the fields you can update.

```
<SSO>
<application name="SSOApplication">
<description>An SSO Application</description>
<contact>someone@companyname.com</contact>
<appUserAccount>DomainName\AppUserGroup</appUserAccount>
<appAdminAccount>DomainName\AppAdminGroup</appAdminAccount>
<flags allowTickets="no" validateTickets="yes"  timeoutTickets="yes" enableApp="no" />
</application>
</SSO>
```

After you create an affiliate application, you cannot modify the following properties:

- Name of the affiliate application.
- Fields associated with the affiliate application.
- Affiliate application type (host group, individual, or configuration store).
- Administration account same as affiliate administrators group. (Specifying this flag will always use the affiliate administrators group as the administrator account for this affiliate application).

Important
You can use local accounts for the administration account and user account by setting the allowLocalAccounts flag to y es. However, you can only use this flag in single-computer scenarios.

Note
You must be an SSO administrator, SSO affiliate administrator, or application administrator to perform this task.

Note
You must be an SSO administrator to modify the ticketing flags (validateTickets and timeoutTickets).

Note
You must be an SSO administrator or an SSO affiliate administrator to modify the application administration account.

To update the properties of an affiliate application using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click the affililate application, and then click **Update**.

To update the properties of an affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory. The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –updateapps <application file name>**, where the *application file name* is the XML file.

See Also

Tasks

How to Enable an Affiliate Application

Concepts

SSO Affiliate Applications

Other Resources

Managing User Mappings

Managing Affiliate Applications

How to Enable an Affiliate Application

You can use the MMC Snap-In or the **enableapp** command to enable the specified affiliate application.

To enable an affiliate application using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click the affililate application, and then click **Enable**.

To enable an affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –enableapp <application name>**, where <*application name*> is the name of the affiliate application you want to enable.

See Also
Tasks
How to Create an Affiliate Application
Concepts
SSO Affiliate Applications
Other Resources
Managing User Mappings
Managing Affiliate Applications

How to Disable an Affiliate Application

Use the MMC Snap-In or the **disableapp** command to disable the specified affiliate application.

To disable an affiliate application using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click the affiliate application, and then click **Disable**.

To disable an affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –disableapp <application name>**, where <*application name*> is the name of the affiliate application you want to disable.

See Also
Tasks
How to Enable an Affiliate Application
Concepts
SSO Affiliate Applications
Other Resources
Managing User Mappings
Managing Affiliate Applications

How to List Affiliate Applications

Use the **listapps** command to list all the affiliate applications. If the user is a member of the Application Administrators account, this command only displays the application for which the user is an administrator.

To list affiliate applications using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On (SSO) installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -listapps [all]** where *all* is an optional parameter that will also display applications using the Configuration Store feature.

   If the user who runs this command is an Application administrator, it only lists the applications for which that user is an administrator. If the user who runs this command is an Affiliate Administrator or an SSO Administrator, it lists all the affiliate applications.

To list affiliate applications using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –listapps** to list the affiliate applications.

   This lists only the affiliate applications that the user who is performing this task is a member of. That is, the user must belong to the application user group account for that affiliate application.

See Also
Tasks
How to Create an Affiliate Application
Concepts
SSO Affiliate Applications
Other Resources
Managing User Mappings
Managing Affiliate Applications

How to List the Properties of an Affiliate Application

The **displayapp** command shows the following information about the affiliate application. For more information about the properties for an affiliate application, see SSO Affiliate Applications.

The Single Sign-On (SSO) system obtains this information from the XML file that you used to update the affiliate application. For more information, see How to Update the Properties of an Affiliate Application.

To display the properties of an affiliate application using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –displayapp <application name>**, where <*application name*> is the name of the affiliate application that you want to display the properties for.

To display the properties of an affiliate application using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*install drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –displayapp <application name>**, where <*application name*> is the name of the affiliate application that you want to display the properties for.

See Also

Other Resources

Managing User Mappings

Managing Affiliate Applications

How to Clear the Application Cache

Use the MMC Snap-In or the **purgecache** command to remove the contents of the credential cache (all the information that is associated with the affiliate application) for the specified application on all Single Sign-On (SSO) servers.

To clear the cache using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Click **Affiliate Applications**.
4. In the results pane, right-click the affiliate application, and click **Clear**.

To clear the cache using the command line

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –purgecache <application name>**, where <*application name*> is the name of the affiliate application that you want to purge the cache for.

See Also
Concepts
SSO Affiliate Applications
Other Resources
Managing Affiliate Applications

How to Set the Enterprise SSO Server Using the Client Utility

Each time you use **ssoclient**, you must first point the user to the correct Single Sign-On (SSO) server that contains their configuration information.

To set the SSO Server for a user using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –server <Single Sign-On Server>**, where <*Single Sign-On Server*> is the name of the Single Sign-On server that the user wants to connect to.

See Also
Concepts
SSO Affiliate Applications
Other Resources
Managing Affiliate Applications

How to Display the Enterprise SSO Server Using the Client Utility

Use the **showserver** command to display the Single Sign-On (SSO) server to which the user is currently pointing.

To display the SSO server using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –showserver**.

See Also

Concepts

SSO Affiliate Applications

Other Resources

Managing Affiliate Applications

How to Set Credentials for the Affiliate Application Using the Client Utility

Use the **setcredentials** command to set the credentials for a user so that the user can access a specific application.

This command does not display the password as you type it.

If the user mapping already exists, this command sets the credentials for the existing mapping. If you have not created the user mapping, the Single Sign-On (SSO) system prompts you for the user ID for the application.

To set credentials for the affiliate application using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –setcredentials <application name>**, where <*application name*> is the specific application for which you want to set the credentials.
4. When prompted for the user credentials, enter the user password for this application.
5. If you have not created the user mapping, the SSO system prompts you for the user ID for the application.

See Also
Concepts
SSO Affiliate Applications
Other Resources
Managing Affiliate Applications

Managing User Mappings

This section provides information about how to create and configure the Enterprise Single Sign-On (SSO) mappings.

Administrators use the **ssomanage** utility to manage mappings, and the application users use the **ssoclient** utility to manage their mappings.

Note
Application administrators can manage mappings associated with affiliate applications for which they are an administrator, while SSO affiliate administrators and the SSO administrators can manage all mappings associated with all affiliate applicati ons.

In This Section

- How to List User Mappings
- How to Create User Mappings
- How to Delete User Mappings
- How to Set Credentials for a User Mapping
- How to Enable a User Mapping
- How to Disable a User Mapping

How to List User Mappings

Use the **listmappings** command to list all the existing mappings for the specified user. You must be a Single Sign-On (SSO) Administrator, Application Administrator, SSO Affiliate Administrator, or user to do this task.

Enabled user mappings appear as (E) <*domain*>\<*username*>, whereas disabled user mappings appear as (D) <*domain*>\<*username*>.

To list user mappings using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. To list all the mappings that a given user has in the affiliate applications that the user belongs to, type:

   ssomanage –listmappings <domain>\<username>

   where <*domain*> is the Windows domain for the user account, and <*username*> is the Windows user name for which you want to list the user mappings. If the user is an Affiliate Administrator or an SSO Administrator, this command lists all the mappings for that user in all the affiliate applications.

   Or
4. To list all the user mappings for a given application, type **ssomanage –listmappings <application name>**.

   Or
5. If you are an application administrator, and you want to list all the mappings a given user has in the affiliate applications for which you are an administrator, type **ssomanage –listmappings <domain>\<username> <application name>**.

To list user mappings using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –listmappings** to list all the mappings you have.

See Also

Tasks

How to Create User Mappings

Concepts

SSO Mappings

Other Resources

Managing Affiliate Applications

Managing User Mappings

How to Create User Mappings

Use the **createmappings** command to create one or more user mappings, as specified in the XML file. The following is an example XML file.

```
<sso>
<mapping>
<windowsDomain>domain</windowsDomain>
<windowsUserId>WindowsUserName</windowsUserId>
<externalApplication>Application name1</externalApplication>
<externalUserId>App1UserName</externalUserId>
</mapping>
<mapping>
<windowsDomain>domain</windowsDomain>
<windowsUserId>WindowsUserName</windowsUserId>
<externalApplication>Application name2</externalApplication>
<externalUserId>App2UserName</externalUserId>
</mapping>
</sso>
```

If a user account is changed, you must use this command to create a mapping for the new user account. You should also remove the old user mapping. For more information about removing a mapping, see How to Delete User Mappings.

After you create a user mapping, you must enable it before you can use this mapping in the Single Sign-On (SSO) system. For more information, see How to Enable a User Mapping.

Important
Only domain groups are supported for user mappings.

To create user mappings using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createmappings <mappings file name>**, where <*mappings file name*> is the name of the file that contains the user mappings that you want to create.

To create user mappings using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –setcredentials <application name >**, where <*application name* > is the name of the affiliate application that the user wants to create a mapping for.

See Also

Concepts

SSO Mappings

Other Resources

Managing Affiliate Applications

Managing User Mappings

How to Delete User Mappings

Use these commands to delete one or more user mappings, as specified in the XML file. The following is an example XML file.

```
<sso>
<mapping>
<windowsDomain>domain</windowsDomain>
<windowsUserId>WindowsUserName</windowsUserId>
<externalApplication>Application name1</externalApplication>
<externalUserId>App1UserName</externalUserId>
</mapping>
<mapping>
<windowsDomain>domain</windowsDomain>
<windowsUserId>WindowsUserName</windowsUserId>
<externalApplication>Application name2</externalApplication>
<externalUserId>App2UserName</externalUserId>
</mapping>
</sso>
```

If a user is not a member of the Application Users account, or does not exist in Active Directory, you should use this command to remove the user mapping from the Credential database.

If a user account is changed, you must use this command to remove the old user mapping, and then create a new user mapping for the new user account. For more information about creating a mapping, see How to Create User Mappings.

To delete user mappings using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –deletemappings <mappings file name>**, where <*mappings file name*> is the name of the file that contains the user mappings that you want to delete.

To delete a specific user mapping using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –deletemapping <domain>\<username> <application name>**, where <*domain*> is the Windows domain for the user account, <*username*> is the Windows user name, and <*application name*> is the specific application for which you want to remove the user mapping.

To delete a user mapping using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –deletemapping <application name>**, where <*application name*> is the name of the affiliate application for which you want to remove the user mapping.

See Also

Concepts

SSO Mappings

Other Resources

Managing Affiliate Applications

Managing User Mappings

How to Set Credentials for a User Mapping

Use the **setcredentials** command to set the credentials for a user to access a specific application.

This command does not display the password as you type it.

If the user mapping already exists, this command sets the credentials for the existing mapping. If you have not created the user mapping, the SSO system prompts you for the user ID for the application.

To set credentials for an user mapping

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –setcredentials <domain>\<username> <applicationname>**, where <*domain*> is the Windows domain for the user account, <*username*> is the Windows user name, and <*applicationname*> is the specific application for which you want to set the credentials.
4. When the SSO system prompts you for the user credentials, enter the user password for this application.
5. If you have not created the user mapping, the SSO system prompts you for the user ID for the application.

To set credentials for a user mapping from client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient -setcredentials <application name>**,where <*application name*> is the name of the affiliate application for which you want to remove the user mapping.

See Also
Tasks
How to Create User Mappings
Concepts
SSO Mappings
Other Resources
Managing Affiliate Applications
Managing User Mappings

How to Enable a User Mapping

You must enable a user mapping before it you can use the mapping in the Single Sign-On (SSO) system.

When you enable a user mapping, it appears as (E) <*domain*>\<*username*> when you list the user mappings.

To enable a user mapping using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –enablemapping <domain>\<username> <application name>**, where <*domain*> is the Windows domain for the user account, <*username*> is the Windows user name for which you want to enable the credentials, and <*application name*> is the name of the affiliate application for which you want to remove the user mapping, and then press ENTER.

To enable a user mapping using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press ENTER.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –enablemapping <application name>**, where <*application name*> is the name of the affiliate application for which you want to remove the user mapping.

See Also

Tasks

How to Create User Mappings

Concepts

SSO Mappings

Other Resources

Managing Affiliate Applications

Managing User Mappings

How to Disable a User Mapping

You can disable a user mapping when you want to turn off all operations associated with a given mapping. You must disable a user mapping before you can remove it.

When you disable a user mapping, it will appear as (D) <*domain*>\<*username*> when you list the user mappings.

To disable a user mapping using the administration utility

1. Click **Start**, click **Run**, type **cmd**, and then press **ENTER**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –disablemapping <domain>\<username><application name>**, where <*domain*> is the Windows domain for the user account, <*username*> is the Windows user name for which you want to disable the credentials, and <*application name*> is the name of the affiliate application for which you want to remove the user mapping.

To disable a user mapping using the client utility

1. Click **Start**, click **Run**, type **cmd**, and then press **ENTER**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default installation directory is <*drive*>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoclient –disablemapping <application name>**, where <*application name*> is the name of the affiliate application for which you want to remove the user mapping.

See Also
Concepts
SSO Mappings
Other Resources
Managing Affiliate Applications
Managing User Mappings

Host Initiated Single Sign-On

Host-initiated Single Sign-On (SSO) enables a request from the host system to access a resource on a Windows system. The host system (for example, an RACF account) exists in a non-Windows environment and under the context of a non-Windows user. The Single Sign-On Credential Store maps host accounts to Windows accounts, enabling this access.

The following topics describe configuration that is specific to Host-initiated SSO.

In This Section

- Configuration Requirements for Host Initiated SSO
- Enabling and Disabling Host Initiated SSO
- Creating Affiliate Applications for Host Initiated SSO
- Validating Passwords for Host Initiated SSO
- Managing User Mappings for Host Initiated SSO
- Troubleshooting Host Initiated SSO

Configuration Requirements for Host Initiated SSO

Although Enterprise Single Sign-On (SSO) and host-initiated SSO have certain aspects in common, certain platform and Active Directory requirements are unique to host-initiated SSO. This topic discusses those requirements, and lists the steps to check or create them on your system.

- Host-initiated SSO can be executed only on a native Windows Server 2003 domain environment.
- The service account for SSO Service that is performing host-initiated SSO must be configured to have Trusted Computing Base (TCB) privileges. (You can configure this for the service account in the domain security policy.)

In addition, certain requirements are necessary when using Transaction Integrator for Host-Initiated Processing (TI for HIP). TI for HIP uses host-initiated SSO to achieve Single Sign-On for non-Windows users.

For example, a service account for TI for HIP service runs under a service account *domainname**hipsvc*. This service can host applications that want to access remote or local resources on Windows with the Windows account that corresponds to the non-Windows account.

The *domainname**hipsvc* account must belong to the Application Administrator group account for the affiliate application that is being used for Single Sign-On.

The *domainname**hipsvc* account must have constrained delegation privileges to use host-initiated Single Sign-On. This can be configured by the domain administrator in Active Directory. Delegation can be configured for accounts that have registered service principal names (SPN). Constrained delegation allows the service account to access only components that are specified by the administrator.

To check your domain function level

1. In the **Active Directory Domains and Trusts** Microsoft Management Console (MMC) snap-in, right-click the **Active Directory Domains and Trusts** node, and then click **Raise Forest Functional Level**.
2. Verify that the functional level is **Windows Server 2003**. If it is not, refer to your Active Directory documentation before you attempt to change the setting.

To create an SPN

1. Download the **setspn** utility from the following location: http://go.microsoft.com/fwlink/?LinkId=148820
2. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
3. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
4. Type **setpsn -a hipsvc\computername.domain.com domain\hissvc**

   where *hipsvc**computername.domain.com* is the service that will perform the operation and the computer it is running on, and *domain**hissvc* is the service account for hipsvc.

After you do this, you can configure constrained delegation in Active Directory for this service account (domain\hissvc) to access the appropriate resource in the network.
To give TCB privileges for the SSO service account

1. Under **Domain Security Policy - Local Policies - User Rights Assignment**, add the SSO Service account to the **Act as part of operating system** policy.

   For more information about Kerberos Protocol Transition and Constrained Delegation, visit http://go.microsoft.com/fwlink/?LinkId=148816.

See Also
Other Resources
Host Initiated Single Sign-On

Enabling and Disabling Host Initiated SSO

By default, host initiated Single Sign-On (SSO) is not enabled in the Single Sign-On system, and must be enabled by the SSO Administrator.

To enable host initiated SSO using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Properties**.
4. Click the **Options** tab.
5. Select the **Enable host initiated SSO** box, and click **OK**.

To enable host initiated SSO using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -enable hisso**.

Disabling SSO applies to the entire SSO system, and all operations related to host initiated SSO are turned off.
To disable host initiated SSO using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **System**, and then click **Properties**.
4. Click the **Options** tab.
5. Clear the **Enable host initiated SSO** box, and click **OK**.

To disable host initiated SSO using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -disable hisso** as appropriate.

See Also
Other Resources
Host Initiated Single Sign-On

Creating Affiliate Applications for Host Initiated SSO

You can define two types of applications:

- **Individual** There is a one-to-one relationship between Windows users and non-Windows users.
- **Host Group** Multiple non-Windows users can be mapped to the same Windows account.

To create an affiliate application using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. Right-click **Affiliate Applications**, and then click **New** to start the **Create New Affiliate Application Wizard**.
4. Use the wizard to select the properties of your affiliate application.

To create an individual type affiliate application using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createapps <AffApp.xml>**, where *AffApp.xml* is the name of the xml file.

   The following is a sample file:

```
<?xml version="1.0"?>
<SSO>
  <application name="SSOApp_Host1">
    <description>An Individual Type Affiliate Application for Host Initiated SSO</desc
ription>
    <contact>someone@companyname.com</contact>
    <appUserAccount>DomainName\AppUserGroup_HISSO</appUserAccount>
    <appAdminAccount>DomainName\AppAdminGroup_HISSO</appAdminAccount>
    <field ordinal="0" label="User ID" masked="no" />
    <field ordinal="1" label="Password" masked="yes" />
    <flags windowsInitiatedSSO="no" enableApp="yes" />
  </application>
</SSO>
```

To create a host group type affiliate application

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createapps <AffApp.xml>**, where *AffApp.xml* is the name of the xml file.

   The following is a sample file:

```
<?xml version="1.0"?>
<SSO>
  <application name="SSOApp_HostGroupApp1">
    <description>A Group Type Affiliate Application for Host Initiated SSO associating
multiple non-Windows user to a single Windows user account(DomainName\WindowsUserAccou
nt1)</description>
```

```
    <contact>someone@companyname.com</contact>
    <windowsAccount>DomainName\WindowsUserAccount1</windowsAccount>
    <appAdminAccount>DomainName\AppAdminGroup_HISSO</appAdminAccount>
    <field ordinal="0" label="User ID" masked="no" />
    <field ordinal="1" label="Password" masked="yes" />
    <flags  enableApp="yes" />
  </application>
</SSO>
```

To create an affiliate application supporting both Windows initiated SSO and host initiated SSO

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createapps <AffApp.xml>**, where **AffApp.xml** is the name of the XML file.

   The following is a sample file:

```
<?xml version="1.0" ?>
- <SSO>
- <application name="SSOApp1">
  <description>An Individual Type Affiliate Application for both Host Initiated SSO an
d Windows Initiated SSO</description>
  <contact>someone@companyname.com</contact>
  <appUserAccount>DomainName\AppUserGroup</appUserAccount>
  <appAdminAccount>DomainName\AppAdminGroup</appAdminAccount>
  <field ordinal="0" label="User ID" masked="no" />
  <field ordinal="1" label="Password" masked="yes" />
  <flags  enableApp="yes" />
  </application>
  </SSO>
```

See Also

Other Resources

Host Initiated Single Sign-On

Validating Passwords for Host Initiated SSO

When an affiliate application for host initiated Single Sign-On (SSO) is created, password validation for the non-Windows user is enabled by default. This means that when applications call SSO to obtain the Windows user token to access resources, they must provide the non-Windows user account and the non-Windows password. If the password does not match the password in the SSO credential database for that non-Windows user, access is denied. If necessary, the password validation feature can be disabled for the affiliate application. The password validation feature applies to both individual and host group type affiliate applications for host initiated SSO.

The following is an example XML file for individual type host initiated SSO affiliate applications:

```
<sso>
<application name="SAP">
<description>The SAP application</description>
<contact>someone@example.com</contact>
<appuserAccount>domain\AppUserGroupAccount</appuserAccount>
<appAdminAccount>domain\AppAdminGroupAccount</appAdminAccount>
<field ordinal="0" label="User Id" masked="no" />
<field ordinal="1" label="Password" masked="yes" />
<flags groupApp="no"  configStoreApp="no" allowTickets="no" validateTickets="yes" allowLoca
lAccounts="no" timeoutTickets="yes" adminAccountSame="no" enableApp="no" />
</application>
</sso>
```

In the case of individual applications for host initiated SSO, the appUserAccount is a group account that contains the list of Windows domain account users that have a one-to-one mapping with their corresponding non-Windows accounts.

The following is an example XML file for host group type host initiated SSO affiliate application:

```
<sso>
<application name="SAP">
<description>The SAP application</description>
<contact>someone@example.com</contact>
<appuserAccount>domain\AppUserAccount</appuserAccount>
<appAdminAccount>domain\AppAdminGroupAccount</appAdminAccount>
<field ordinal="0" label="User Id" masked="no" />
<field ordinal="1" label="Password" masked="yes" />
<flags  configStoreApp="no" allowTickets="no" validateTickets="yes" allowLocalAccounts="no"
timeoutTickets="yes" adminAccountSame="no" enableApp="no" />
</application>
</sso>
```

In group applications for host initiated SSO, the appUserAccount must be an individual user account. It is this account that all non-Windows accounts are mapped to.

See Also

Other Resources

Host Initiated Single Sign-On

Managing User Mappings for Host Initiated SSO

Use the following procedures to create mappings, set credentials (whenever the Validate Password feature is enabled for the affiliate application), and enable/disable mapping.

To manage user mappings for host initiated SSO using the MMC Snap-In

1. Click **Start**, point to **Programs**, click **Microsoft Enterprise Single Sign-On**, and then click **SSO Administration**.
2. In the scope pane of the ENTSSO MMC Snap-In, expand the **Enterprise Single Sign-On** node.
3. In the scope pane, click **Affiliate Applications**.
4. In the details pane, right-click the affiliate application, and then select the appropriate menu item for your action.

To create mappings in host initiated SSO using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage –createmappings <mapping file>**, where <*mapping file*> is the name of the XML file.

   The following is a sample mapping file:

```
<SSO>
  <mapping>
    <windowsDomain>DomainName</windowsDomain>
    <windowsUserId>UserA</windowsUserId>
    <externalApplication>SSOApplication</externalApplication>
<externalUserId>ExternalUserID that corresponds to UserA</externalUserId>
  </mapping>
</SSO>
```

To set credentials for individual type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory. The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -setcredentials <Windows account name> <application name>**.

To set credentials for host group type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command line, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -setcredentials <external account name> <application name>**.

To enable mappings for individual type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -enablemapping <Windows account name> <application name>**.

To disable mappings for individual type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -disablemapping <Windows account name> <application name>**.

To enable mappings for host group type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -enablemapping <external account name> <application name>**.

To enable mappings for individual type affiliate applications using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -disablemapping <external account name> <application name>**.

See Also

Other Resources

Host Initiated Single Sign-On

Troubleshooting Host Initiated SSO

The primary method of troubleshooting host initiated Single Sign-On (SSO) is tracing.

Tracing

Use the Trace command-line utility to enable tracing in SSO.

Note
For the trace command to function, the file tracelog.exe must be in the following directory: \<drive\>:\Program Files\Common Files\Enterprise Single Sign-On. You can download this file from the following location: http://go.microsoft.com/fwlink/?LinkId=33023

To use the trace utility

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **Trace –start –high** to set the tracing level to high and begin the trace.
4. Run the scenario with host initiated SSO.
5. Type **Trace –stop** to end the trace.
6. A .bin file is generated in the directory above, which you can send to Microsoft for analysis.

See Also

Other Resources

Host Initiated Single Sign-On

How to Use the ENTSSO Management Agent

This version of Enterprise Single Sign-On (SSO) contains a Management Agent (MA) for Microsoft Identity Integration Server (MIIS), which integrates Enterprise SSO with the account synchronization capabilities of MIIS. This enables an MIIS administrator to manage SSO mappings in the SSO Credential Database.

In Enterprise SSO, mappings are created between Windows Domain accounts (*domainname**username*) and external credentials. If you have an Active Directory Management Agent, and the Management Agent for the external Data Source (example: RACF MA), you can use the Enterprise SSO MA (ENTSSO MA) to manage mappings in the SSO Credential Database. ENTSSO MA is a *Call-Based Export* Management Agent only.

You configure the Management Agent in three separate parts:

- A configuration file (ENTSSO.xml)
- The MIIS user interface
- The ENTSSO user interface

The topics in this section describe the configuration process.

In This Section

How to Configure the XML File

How to Configure MIIS for ENTSSO MA

How to Configure ENTSSO for MIIS Password Sync

See Also

Other Resources

Security User's Guide

How to Configure the XML File

When you install Enterprise Single Sign-On (SSO), an XML file named ENTSSO.xml is installed in your Extensions directory. By editing the file, you define the configuration for all instances of the ENTSSO Management Agent (MA).

The file is similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<sso>

  <SourceMA name="RACFMA1" objectType="person" attribute="uid"/>
  <SourceMA name="ACF2" objectType="person" attribute="uid"/>

  <ENTSSOMA name ="ENTSSOMA1" adma="ADMA1" deleteAll="true">
    <Application name="AppForRACF1A" sourceMA="RACFMA1" create="true" delete="true"/>
    <Application name="AppForRACF1B" sourceMA="RACFMA1" create="true" delete="false"/>
  </ENTSSOMA>

  <ENTSSOMA name ="ENTSSOMA2" adma="ADMA1" deleteAll="true">
    <Application name="AppForACF2" sourceMA="ACF2"/>
  </ENTSSOMA>

</sso>
```

XML Elements and Attributes

The following list describes the elements and attributes that you define in the XML file. Note that all element and attribute names in this file are case sensitive.

Element: ENTSSO - Defines the configuration of a single ENTSSO MA instance. Multiple ENTSSO elements are allowed.

Attribute: name - Defines the instance name of the ENTSSO Management Agent, and must match the name of the ENTSSO Management Agent instance in Microsoft Identity Integration Server (MIIS).

Attribute: adma - Defines the instance name of the Active Directory Management Agent that will be used by this ENTSSO Management Agent instance. The Active Directory Management Agent provides the Windows domain name and Windows user name for the mapping. This Active Directory Management Agent instance name must match the name of an Active Directory Management Agent instance in MIIS.

Attribute: deleteAll - Optional; default is **true**. If this is set to **true**, and a Windows identity is deleted, all mappings with that Windows identity are deleted from all ENTSSO applications.

Element: Application - Defines the relationship between an SSO affiliate application and an external Management Agent. Multiple **Application** elements are allowed.

Attribute: name - Defines the name of the SSO affiliate application. This application must already exist within the ENTSSO system.

Attribute: sourceMA - Defines the instance name for the source (external) Management Agent that will be used to provide the external UserId in the mapping for this application. This external Management Agent instance name must match the name of an external MA instance in MIIS.

Attribute: create - Optional; default is **true**. Defines whether mappings should be created for this application.

Attribute: delete - Optional; default is **true**. Defines whether mappings should be deleted for this application.

Element: SourceMA - Optional. Identifies the object type and attribute names for a specific source (external) Management Agent instance. If this element is not present for a specific Management Agent, then the default object type ("person") and attribute names ("uid") are assumed. Multiple **SourceMA** elements are allowed.

Attribute: name - The name of the source (external) Management Agent. This name must match at least one of the **sourceMA** attribute names from the **Application** elements.

Attribute: objectType - Optional; default is **person**. If the object type name that provides the external UserId is not **person**, it should be specified here.

Attribute: attribute - Optional; default is **uid**. If the attribute name that provides the external UserId is not **uid**, you can specify it here.

See Also

Concepts

How to Use the ENTSSO Management Agent

Other Resources

Security User's Guide

How to Configure MIIS for ENTSSO MA

When you install the Enterprise Single Sign-On (SSO) Administration feature (either the full version or the Admin-only version) on a computer running Microsoft Identity Integration Server (MIIS), the ENTSSO Management Agent is automatically installed. This means that when you open MIIS, nearly all of the configuration has already been done. The only part missing is the connection information.

Before starting this procedure, make sure you have the following information available:

- ENTSSO Server name.
- UserId and password of the Windows account under which the ENTSSO Management Agent will communicate with the SSO Server.

To configure the Management Agent within MIIS

1. Open MIIS, and open the **Identity Manager**.
2. Open the **Create Management Agent** dialog box.
3. Select **Enterprise Single Sign-On** in the list.

   This starts the **Create Management Agent Wizard**.
4. On the **Configure Connection Information** page, in the **Connect To:** field, enter the name of the SSO Server.
5. Enter the name of the ENTSSO Management Agent. This name must match the name specified in your ENTSSO.xml file.
6. In the **User** field, specify the domain account that the ENTSSO Management Agent uses to manage mappings in the SSO Credential Database.

   This account should be either a member of the SSO Affiliate Administrators or SSO Administrators accounts within the SSO System.
7. In the **Password** field, enter the password for that user.
8. Click **Next**, accepting the defaults until you reach the **Configure Extensions** page.
9. Near **Connection information** for password extension, click **Settings**.

   The **Connection Settings** dialog box appears.
10. In the **Connect To** box, enter the appropriate account. This account must be the same as the service account for the ENTSSO service running on the computer specified.
11. In the **User** and **Password** fields, enter the user name and password for the account.
12. Click **OK**.
13. In the **MIIS Create Management Agent**, click **Finish**.
14. While still in the **Identity Manager**, click the **Tools** menu, and then click **Options**.

    The **Options** dialog box appears.
15. Select **Enable metaverse rules extension**.
16. In the **Rules extension name field**, enter **Microsoft.EnterpriseSingleSignOn.ManagementAgent.dll**.
17. Click **OK** and close MIIS.

Example

If you already have a Metaverse rules extension that you want to use, you can copy the following code example and edit it appropriately.

```
// <copyright file="MVWrapper.cs" company="Microsoft">
// Microsoft Enterprise Single Sign-On
```

```
// Copyright (c) Microsoft Corporation.  All rights reserved.
// </copyright>

using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.EnterpriseSingleSignOn.ManagementAgent;
using Microsoft.MetadirectoryServices;
using System.Diagnostics;

// This sample code illustrates how to call the Enterprise
// Single Sign-On (ENTSSO) MV rules
// extension from your own MV rules extension.
// A reference is required to
// Microsoft.EnterpriseSingleSignOn.ManagementAgent.dll.

namespace MVWrapper
{
    public class MVWrapper : IMVSynchronization
    {
        MVSync mvSync = null;

        public void Initialize()
        {
            Debug.WriteLine("IMVSynchronization.Initialize");

            mvSync = new MVSync();

            mvSync.Initialize();
        }

        public void Provision(MVEntry mventry)
        {
            Debug.WriteLine("IMVSynchronization.Provision");

            mvSync.Provision(mventry);
        }

        public bool ShouldDeleteFromMV(CSEntry csentry, MVEntry mventry)
        {
            Debug.WriteLine("IMVSynchronization.ShouldDeleteFromMV");

            return mvSync.ShouldDeleteFromMV(csentry, mventry);
        }

        public void Terminate()
        {
            Debug.WriteLine("IMVSynchronization.Terminate");

            mvSync.Terminate();

            mvSync = null;
        }
    }
}
```

See Also
Concepts
How to Use the ENTSSO Management Agent

How to Configure ENTSSO for MIIS Password Sync

After configuring the XML file and Microsoft Identity Integration Server (MIIS), the remaining configuration steps take place in the Enterprise Single Sign-On (ENTSSO) system.

To allow Password Sync from MIIS

1. In Enterprise Single Sign-On, click the **Servers** node.
2. Right-click the appropriate server, and click **Properties**.
3. Click the **Password Sync** tab.
4. Select **Allow password sync from MIIS**.
5. Click **OK**.

To enable Password Sync on the system level

1. In Enterprise Single Sign-On, right-click the **System** node.
2. Click **Properties**.

   The **Properties** dialog box appears.
3. Click the **Options** tab.
4. In the **Enable Password Sync** field, select **From Windows to Adapters**.

Additional Configuration

Finally, you must configure one of the following:

- A Password Sync Adapter that accepts Windows Password Sync.
- Direct Password Sync enabled on at least one application.

For information about how to do this, refer to your Password Sync documentation.

To configure the EntSSO MA for MIIS Password Sync

1. On the ENTSSO Management Agent **Properties** page, click **Configure Extensions**.
2. In the **Connection information for password extension** field, click **Settings**.
3. In the **Connect To** field enter the name of the computer that will receive the password changes.

   The computer name must be in the same format that was used when creating the Service Principal Name (SPN) for the ENTSSO service on the domain.

   For example:

   Short format - SPN = ENTSSO/ABCD1411, then enter ABCD1411

   Long format - SPN = ENTSSO/ABCD1411.CompanyName.com then enter ABCD1411.CompanyName.com

Additional Configuration Steps

1. Click **Start**, point to **All Programs**, point to **Microsoft Identity Integration Server**, and then click **Identity Manager**.
2. On the **Tools** menu, click **Options**.
3. Select **Enable Password Synchronization**.
4. In the **Management Agents view**, select **ADMA**.
5. In the **Action** pane, select **Properties**.
6. On the **Properties** page, select **Configure Directory Partitions**, and then select **Enable this partition as a password synchronization source**.

7. Click **Targets**, and then select ENTSSOMA2 to enable it to receive password changes from MIIS. Deselect ENTSSOMA. Click **OK**, and then click **OK** again.
8. In the **Management Agent** view, select **ENTSSOMA2**. In the right-hand pane, select **Properties**. On the **Properties** page, click **Configure Extensions**.
9. Confirm that **Enable password management** is selected, and then click **Settings**.
10. In the **Connection Settings** dialog, specify the following:
    - Connect To: INTSVR1.fabrikam.com
    - User: fabrikam\ssosvcact
    - Password: ssosvcact

Note
This account should match the ENTSSO service account configured on INTSVR1.fabrikam.com.

11. Click **OK**, and then click **OK** again.
12. You can also disable password sync for MIIS. To do this, in **Identity Manager**, click the **Tools** menu, click **Options**, and then deselect **Enable Password Synchronization**.

    The following restrictions will apply:

    - For Password Sync to function properly, SPN must be configured on the ENTSSO service account that the ENTSSO Management Agent will communicate with.
    - Communication between MIIS and the ENTSSO server requires Kerberos.
    - When configuring Password Extension in the MIIS connection configuration for the ENTSSO Management Agent, the account specified must match the service account for the ENTSSO server that will receive passwords from MIIS.

See Also

Concepts

How to Use the ENTSSO Management Agent

How to Use the Servers Snap-In

This version of Enterprise Single Sign-On (SSO) includes the ENTSSO Servers Snap-In, which allows you to view, monitor, and perform certain actions on your ENTSSO Server through the Windows interface.

Note
If your system has a firewall and your server name uses the FQDN format, you may not be able to contact the server. Instead, you must specify the NetBIOS name or the associated IP address.

To use the ENTSSO Servers Snap-In

1. Open Enterprise Single Sign-On.
2. In the Scope pane, click the **Servers** node.

   The following information is displayed in the Results pane.

   - **Name**: Name specified.
   - **Status**: Status of the ENTSSO service (Online, Offline, Pending, Started, Stopped, Start Pending, Stop Pending).
   - **Date**: Date when information was obtained.
   - **Time**: Time when information was obtained.
   - **SSO Server**: Fully qualified name of server.
   - **Service Account**: ENTSSO service account.
   - **SSO Database**: ENTSSO Credential Database with which this server is communicating.
   - **Secret Server**: Indicates whether the Secret Server module is running.
   - **Password Sync**: Indicates whether Password Sync is installed.
   - **Computer**: NETBIOS name of computer.
   - **Event counts**: Info/Warning/Errors event count. Resetting this will start the counter from 0.
   - **Version of SSO installed**: Version number of ENTSSO.exe. Also indicates whether this is 32-bit (x86) or 64-bit (x64).

To start or stop the server service

- In the ENTSSO Servers Snap-In, right-click the server and click **Start** or **Stop**.

To display information for one server

- In the Server tree, click the server.

  The information is displayed in the results pane.

To add a server

- Right-click in the Scope pane, and then click **Add Server**.

  The **Add Server** dialog box appears. Type or browse to the server you want to add.

  In certain Workgroup environments, clicking the **Browse** button will cause this dialog box to close. Instead of using the **Browse** button, simply type the server name in the box.

To view or change server properties

- In the Server tree, right-click the server, and click **Properties**.

  The **Server Properties** dialog box appears. You can view or change information in the following tabs:

  - **Audit Levels**
  - **SSO Database**
  - **SSO Service**
  - **Password Sync**
  - **Advanced**

See Also

Other Resources

Security User's Guide

How to Use Direct Password Sync

This version of Enterprise Single Sign-On includes the Direct Password Sync from Windows feature. This enables you to bypass a Password Sync Adapter and update the password in the ENTSSO Credential Database directly from Windows.

Direct Password Sync from Windows is useful in the following situations:

- Your enterprise system requires Windows to Windows mapping.
- You need to update the External User's password in the Credential Database directly when a password change occurs for the Windows user. You can change the password on the back-end system (that corresponds to the external user) by using other mechanisms. For example, you can use Microsoft Identity Integration Server to update passwords in Resource Access Control Facility (RACF) on an IBM Mainframe using the RACF Management Agent.

To enable Direct Password Sync

1. Open Enterprise Single Sign-On.
2. In the scope pane, click **Affiliate Applications**.
3. Right-click the appropriate **Affiliate Application**.
4. Click **Properties**.

   The **Affiliate Applications Properties** dialog box appears.
5. Click the **Options** tab.
6. Select the **Direct Password Sync from Windows** check box.
7. Click **OK**.

How to Use the Mapping Wizard

This version of Enterprise Single Sign-On includes the Mapping Wizard, which allows you to easily create mappings for affiliate applications.

Note
You can only use the Mapping Wizard if the External UserID is based on the Windows domain account.

To start the Mapping Wizard

1. Open Enterprise Single Sign-On.
2. In the scope pane, click **Affiliate Applications**.
3. Right click the appropriate affiliate application.
4. Click **Create Mappings**.

   The Mapping Wizard appears.

To use the Mapping Wizard

1. **Welcome to the Mapping Wizard**
   - Verify that this is the correct affiliate application, and click **Next**.
2. **Mappings File Option** page
   - ENTSSO manages mappings through an XML file. You can choose to create a new file or use an existing one.
3. **Files Location** page
   - Select the files to be used.
   - You must click **Validate** to validate the files before proceeding. The validation status appears in the **Status** window.
4. **Accounts** page
   - Choose one or more accounts to generate the new mappings file, and click **Validate**.
5. **External User Name** page
   - Define how the external user name is generated from the Windows user account. As you make selections in the boxes, you can see how the names will appear in the **Example** box.
6. **Generate** page
   - Click **Start** to generate the mappings file. (You will have an opportunity on the next page to view or edit the file as necessary before mappings are created.) Results are displayed in the three windows below.
   - The **Number** of mappings in selected accounts is an approximation, because accounts may be nested in other accounts.
   - Click **View Log File** to examine any errors or warnings that might have occurred.
7. **Options** page

- Your file has been created. Click **View/Edit Mappings File** if desired.
- Click **Enable mappings** if you want the mappings to be automatically enabled. (If you do not click this, you will have to enable them manually.)
- Click **Set Password** to automatically set the password for these mappings.

8. **Create** page
   - Before creating the mappings, click **View Mappings File** if desired.
   - When you are ready, click **Start** to perform the mapping operation. Your status is displayed in the three boxes below.
   - Click **View Log File** to examine any errors or warnings that might have occurred.

9. **Completing the Mapping Wizard screen**
   - Select any options desired, and then click **Finish**.

See Also
Concepts
SSO Security Recommendations
Other Resources
Using Enterprise Single Sign-On

How to Use Password Filters

The ENTSSO Password Synchronization feature synchronizes passwords between Microsoft Windows Active Directory and non-Windows systems. However, many external systems have password policy requirements which differ from those in Active Directory. (For example, an IBM system may require a password to be upper case and limited to 8 characters.) This forces ENTSSO to use the "lowest common denominator" between the two systems, limiting password security.

The ENTSSO Password Filter feature addresses this limitation. A Password Filter is merely a Password Sync Adapter with additional properties defined. These additional properties (such as maximum or minimum length, case, and character restrictions) serve to filter the passwords so that they meet the criteria of the external system.

Note that when you create a Password Filter, the Administrator group specified is automatically used as the SSO Administrators account. If you change the SSO Administrator group, you will need to make sure the Password Filter is also updated with the new SSO Administrators account.

Note
As with all elements of the ENTSSO system, Password Filters contain highly sensitive information and should be exposed to t he minimum number of people possible.

To Create a Password Filter

1. In the **SSO Management Console**, right-click the **Password Management** node, and then click **Create Filter**.

   The **Password Filter Wizard** appears.

2. Follow the directions on the Wizard to create the Password Filter.

To Assign an Affiliate Application to a Password Filter

1. Right-click the appropriate filter, and then click **Assign**....
2. Select the appropriate **Affiliate Application**.

Secure Deployment of Enterprise Single Sign-On

This section outlines a typical scenario for secure deployment of Enterprise Single Sign-On (SSO). For detailed procedures on the actions to take in SQL Server, see your SQL Server documentation.

In This Section

Deployment Overview

The system in this example is deployed over three domains, and contains the following computers:

Domain ORCH.com

- ORCH domain controller
- HIS1, the HISSO server
- HIS2, the master secret server
- HIS3, the Admin database

Domain SQL.com

- SQL domain controller
- SQL2, the SSO database

Domain HIS.com

- HIS domain controller
- HIS4 database

The key points defining this deployment are as follows:

- Domain ORCH.com and domain SQL.com have a two-way selective trust relationship.
- Domain ORCH.com is configured as native Windows Server 2003 server functional level.
- All SSO services are running on an ORCH.com domain user account (Orch\SSOSvcUser). The user is configured to have access permission on the SQL2 machine in the SQL.com domain. The user is configured for protocol transition and constrain delegation within the ORCH.com domain.
- Another ORCH.com domain user (Orch\TestAppUser) is set for running test programs. This user is also configured for protocol transition and constrain delegation.

For a description of the deployment process, see Deployment Process

See Also
Concepts
Deployment Process
Other Resources
Secure Deployment of Enterprise Single Sign-On

Deployment Process

The following steps give a high-level overview of secure deployment of Enterprise Single Sign-On (SSO). For detailed procedures on the actions to take in SQL Server, see the SQL Server documentation.

1. On the SQL Server domain controller, use the **New Trust Wizard** to create a trust with the following properties:

    - **Name:** ORCH.com
    - **Direction:** Two-way
    - **Sides:** This domain only
    - **Outgoing Trust Authentication Level - Local Domain:** Selective authentication
    - **Password:** Choose a password
    - **Confirm Outgoing Trust:** Yes
    - **Confirm Incoming Trust:** No

2. On the ORCH.com domain controller, use the **New Trust Wizard** to create a trust with the following properties:

    - **Name:** SQL.com
    - **Direction:** Two-way
    - **Sides:** This domain only
    - **Outgoing Trust Authentication Level - Local Domain:** Selective authentication
    - **Password:** Must be the same as password for ORCH.com
    - **Confirm Outgoing Trust:** Yes
    - **Confirm Incoming Trust:** No

3. On the ORCH.com domain controller, set the domain-wide trust for Incoming from SQL.COM.
4. On the SQL.com domain controller, set the domain-wide trust for Outgoing from ORCH.COM.
5. On the ORCH.com domain controller, raise the domain functional level to Windows Server 2003.
6. In the ORCH domain, create the following new users:

    - ORCH\SSOSvcUser
    - ORCH\TestAppUser
    - ORCH\AffAppUser

7. Add **Act as part of the operating system** to SSOSvcUser and TestAppUser.

8. Add **Allowed to Authenticate** privilege to ORCH\TestAdmin.

9. Add ORCH\SSOSvcUser to SQL2 in the SQL domain. This step requires using Advanced View in Active Directory Microsoft Management Console (MMC).

10. On the SQL2 computer, create the following two new logons:

    - ORCH\TestAdmin
    - ORCH\SSOSvcUser

11. On the SQL2 domain, create two domain global groups:

    - ORCH\SSOAdminGroup
    - ORCH\SSOAffAdminGroup

12. Add **Allowed to Authenticate** privilege to the ORCH\SSOAdminGroup group.

13. On the SQL2 database, create the following new logon:

    - ORCH\SSOAdminGroup

14. Install the master secret server as follows:

    - Log onto NTS5 using ORCH\TestAdmin.
    - Install Enterprise SSO, using SQL2 as the master secret server.

15. Log on to HIS1 using ORCH\TestAdmin, and install Enterprise Single Sign-On. Configure ESSO as SSO join HIS2, using database server name SQL2.

16. Install the Enterprise Single Sign-On Admin utility on HIS3 using ORCH\TestAdmin.

17. Add the following users to the following groups:

    - Add ORCH\TestAppUser to ORCH\SSOAdminGroup
    - Add ORCH\AffAppUser to ORCH\TestAffUserGroup

18. Install SQL Server 2000a Enterprise on HIS3, and add logon ORCH\AffAppUser.

19. On the HIS1 machine, open a command prompt and use the following commands to set constrain delegation and protocol transition:

    - **setspn -A MSSQLSvc/HIS3.ORCH.com:1433 ORCH\SSOSvcUser**
    - **setspn -A MSSQLSvc/HIS3.ORCH.com:1433 ORCH\TestAppUser**

20. On the **ORCH\SSOSvcUser** and **ORCH\TestAppUser** property pages, set the proper delegation for both user accounts by selecting the following options:

    - **Trust this user for delegation to specified services only**
    - **Use any authentication protocol**

21. Using ORCH\TestAdmin on the HIS1 computer, perform the following:

    - Add ORCH\TestAppUser to Remote Desktop User Group.
    - Grant **Impersonate after authenticated** privilege to ORCH\SSOSvcUser.
    - Grant **Impersonate after authenticated** privilege to ORCH\TestAppUser.

22. Verify your deployment by logging on to HIS1 using ORCH\TestAppUser and running the following application configuration:

    Run LogonExternalUser Test.

```
<SSO>
   <application name="TestApp">
      <description>An SSO Test Affiliate Application</description>
      <contact>AffAppUser@ESSOV2.EBiz.Com</contact>
      <appUserAccount>ORCH\TestAffAdminGroup</appUserAccount>
      <appAdminAccount>ORCH\TestAffUserGroup</appAdminAccount>
      <field ordinal="0" label="User ID" masked="no" />
      <field ordinal="1" label="Password" masked="yes" />
      <flags
         groupApp="no"
         configStoreApp="no"
         allowTickets="no"
         validateTickets="yes"
         allowLocalGroups="yes"
         ticketTimeout="yes"
         adminGroupSame="no"
         enableApp="yes"
         hostInitiatedSSO="yes"
         validatePassword="yes"/>
   </application>
</SSO>
```

See Also

Concepts

Deployment Overview

Password Synchronization

The purpose of Password Synchronization is to simplify administration of the Single Sign-On (SSO) credential database, and to keep passwords in sync across user directories. You can accomplish these two tasks by using password synchronization adapters. The topics in this section describe the command-line utility for creating and managing those adapters.

There are three types of password synchronization sub-features.

The first type is **Windows to External** (for example, Active Directory to RACF). In this scenario, a Windows user's password change is captured and sent to the Enterprise SSO server that is assigned to receive password changes from domain controllers. This server then forwards the password change to an external system, and the mapping in the SSO credential database is kept in sync with the change made on the external system.

The second type is **External to Windows - Full synchronization**. In this scenario, a password is captured on the External system and sent to the Enterprise Single Sign-On server that is assigned for Password Synchronization. It then updates the password in the SSO credential database, and also updates the Windows user's password in Active Directory.

The third type is **External to Windows - Partial synchronization**. In this scenario, a password is captured on the External system and sent to the Enterprise Single Sign-On server that is assigned for Password Synchronization. It then updates the password in the SSO credential database.

In This Section

Installing Password Synchronization

As with the other Single Sign-On (SSO) features, Password Synchronization is not installed in the default Host Integration Server installation, and must be specifically selected during Setup.

To install Password Synchronization

1. Run the Host Integration Server Setup program, and select **Custom Installation**.
2. Under the **Security Integration** feature, select the **Enterprise Single Sign-On Password Synchronization** sub feature.
3. Complete the installation.

You also need Password Synchronization adapters to send and receive password changes to the external system. Other topics in this section describe how to configure your own adapters. For more information, see Administering Password Synchronization. You can also view a list of currently available adapters at the following location: http://go.microsoft.com/fwlink/?LinkId=68145. You can contact support aliases to obtain information about these Password Synchronization adapters.

Finally, to capture password changes made in Active Directory, in addition to installing the ENTSSO Password Sync feature, you must install components on the domain controllers to capture password changes. Both the Windows Password Capture component and Password Change Notification Service (PCNS) must be installed on all domain controllers from which you will be capturing passwords. You can find these components in the following location:

http://www.microsoft.com/downloads/details.aspx?FamilyID=C0964F2E-FA9F-4FC7-AC13-C43928EFEE9D&displaylang=en

Read the accompanying documentation before you proceed with the installation on the domain controller.

See Also
Other Resources
Password Synchronization

Administering Password Synchronization

You can administer Password Synchronization in Enterprise Single Sign-On (SSO) through either the Microsoft Management Console (MMC) Snap-In or the command line. This topic describes how to perform various administration tasks with adapters.

The MMC Snap-In displays a list of adapters and their properties. You can right-click an adapter and use the menu to perform the following commands:

- Create adapters
- Set properties
- Update
- Delete
- Enable
- Disable
- Add applications to an adapter
- Delete applications from an adapter
- Reset notification
- Add an adapter to an adapter group
- Delete an adapter from an adapter group

You can also use the SSOPS command-line utility to administer your password synchronization. Most of the commands in this section are intended for use by an administrator only.

For many commands, the command output is displayed on the screen in two columns. Because certain screen settings could cause truncation of data, for best results you should change the screen buffer size/Windows size to 120 characters.

The following table lists the SSOPS commands. Procedures and additional explanation are located throughout the rest of this topic.

Command	**Function**
-list	Lists existing adapters.
-display	Displays adapter information.
-create	Creates new adapters.
-setprops	Sets properties for adapter.
-update	Updates existing adapters.
-delete	Deletes an existing adapter.
-enable	Enables adapter.
-disable	Disables adapter.

-addapp	Adds application for adapter.
-deleteapp	Deletes application for adapter.
-reset	Resets notification or damping queues.
-addtogroup	Adds adapter to adapter group.
-deletefromgroup	Deletes adapter from adapter group.

To list existing adapters

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -list** and press ENTER.

   Adapters and descriptions are listed. (E) denotes that the adapter is enabled, (D) denotes that it is disabled.

To display adapter information

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -display <adapter name>** and press ENTER.

   The screen output displays information for the specified adapter.

   In addition to name, type, description, computer, and accounts, the following information is displayed.

Adapter Flag	**Details**
Adapter enabled	Determines whether the adapter is enabled. Flag: SSO_FLAG_ENABLED Attribute Name: enableApp Default: No
Allow local accounts	Determines whether the App Admin or App Users accounts can be local accounts. Flag: SSO_FLAG_APP_ALLOW_LOCAL Attribute Name: allowLocalAccounts Default: No
Receive password changes from adapter	Determines whether the adapter is allowed to receive external password changes. Flag: SSO_FLAG_PARTIAL_SYNC_FROM_EXTERNAL_TO_DB Attribute Name: syncFromAdapter Default: No

Verify old password	Determines whether the adapter will verify the old password when an external password change is received. If this flag is set, when an external password change is received, the external adapter must supply the old external password in addition to the new external password. The old external password is then compared with the existing external password in the SSO database for that external account. If they match, the password change is accepted. If they do not match, the password change is rejected. Flag: SSO_FLAG_SYNC_VERIFY_EXTERNAL_CREDS Attribute Name: verifyOldPassword Default: Yes
Change Windows password	Determines whether the Windows password will also be changed when an external password change is received (full sync). ENTSSO always uses the old Windows password stored in the SSO database to change the Windows password to the new value (Windows requires both the old and new password to change a user's password). Therefore, this must be initialized before the Windows password change can succeed. If password sync is configured for a particular mapping, when the external credentials are set via administrative tools (ssomanage or ssoclient -setcredentials), the Windows password that is stored in the SSO database is also set. Flag: SSO_FLAG_FULL_SYNC_FROM_EXTERNAL_TO_WINDOWS Attribute Name: changeWindowsPassword Default: No
Send Windows password changes to adapter	Determines whether Windows password changes are sent to the external adapter. Flag: SSO_FLAG_FULL_SYNC_FROM_WINDOWS_TO_EXTERNAL Attribute Name: syncToAdapter Default: No
Send old password to adapter	If Yes, the old password value (from the SSO database) is also sent to the external adapter in addition to the new password value. Some external systems might require both the old and new password values to change the password. Flag: SSO_FLAG_SYNC_PROVIDE_OLD_EXTERNAL_CREDS Attribute Name: sendOldPassword Default: No
Allow mapping conflicts	Determines whether the adapter will allow mapping conflicts. A mapping conflict occurs when mappings are not unique. In a single SSO Individual application, mappings are always one-to-one: one Windows account is mapped to exactly one external account and vice versa. However, it is possible to assign more than one application to an adapter. Thus, it is possible to have a mapping in one application that conflicts with a mapping in the other. The purpose of this flag is to prevent this from occurring. It is more secure to not allow mapping conflicts unless there is a specific, well understood requirement for this behavior. Flag: SSO_FLAG_SYNC_ALLOW_MAPPING_CONFLICTS Attribute Name: allowMappingConflicts Default: No

Adapter Description	**Details**
Notification retry count	Default is 1.
Notification retry delay (in mins)	Default is 5.

Maximum pending notifications	Default is 8.
Store notifications (when offline)	True/False.
Server name	Server name.
Port number	Port number.
Applications for this adapter	List of applications currently assigned to the adapter.

To create new adapters

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -create <adapter file>** and press ENTER.

   The screen output displays information for the newly created adapter.

To set properties for an adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -setprops <adapter name>** and press ENTER.

   The screen output displays the properties for the specified adapter. You can edit them if necessary, but new values are not validated.

To update existing adapters

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -update <adapter file>** and press ENTER.

   Use this command to update the settings and flags for a specified adapter. Do not use this command to set properties; use the **-setprops** command instead.

To delete an existing adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -delete <adapter name>** and press ENTER.

   The specified adapter is deleted.

To enable an adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.

3. Type **ssops -enable <adapter name>** and press ENTER.

   The specified adapter is enabled.

To disable an adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -disable <adapter name>** and press ENTER.

   The specified adapter is disabled.

To add an application to an adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command line, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -addapp <adapter name> <application name>** and press ENTER.

   The specified SSO application is assigned to the specified adapter. This means that the passwords for the mappings in that application are now synchronized using this adapter.

   Although multiple applications can be assigned to one adapter, any given application can only be assigned to one adapter.

To delete an application from an adapter

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -deleteapp <application name>** and press ENTER.

   The specified SSO application is removed from an adapter. (Because an application can only be assigned to one adapter, you do not have to specify the adapter name.)

To reset notification

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -reset <adapter name | all | damping>** and press ENTER.

   This command clears the damping table and/or notification queues for a single adapter or all adapters, as specified. The damping table stores a 10-minute history of password changes. Before the Enterprise SSO system accepts or sends a password change, it checks the damping table to see whether it has performed the same change recently. If it has, the new change is discarded.

To add an adapter to an adapter group

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -addtogroup <adapter name> <adapter group>** and press ENTER.

   This command adds the specified adapter to the specified adapter group. Although an adapter can belong to only one adapter group, an adapter group can contain multiple adapters.

To delete an adapter from an adapter group

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssops -deletefromgroup <adapter name> <adapter group>** and press ENTER.

   This command deletes the specified adapter from the specified adapter group.

See Also

Other Resources

Password Synchronization

Configuring Password Synchronization

Use the SSOCONFIG command-line utility to configure your password synchronization settings. You can use this tool to specify directories for replay files and also change maximum password synchronization age.

To specify the directory for replay files

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoconfig -replayfiles <replay files directory> | -default** and press ENTER.

Note Replay files are not deleted when you change the service account. If you change this account, you must delete the replay files manually.

To display or change maximum password synchronization age

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssoconfig -syncage <maximum password age in hours>** and press ENTER.

Note
The SSOCONFIG utility uses the time on the computer that is running SQL Server as its system time. Remember this when you are using any commands related to time.

See Also

Other Resources

Password Synchronization

Managing Password Synchronization

Use the SSOMANAGE command-line utility to enable or disable Single Sign-On (SSO) features, and to display current SSO database settings.

To manage features or display settings using the MMC Snap-In

1. Right-click the appropriate feature or database.
2. Click the appropriate command.

To enable SSO features using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -enable** and press ENTER.

To disable SSO features using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -disable** and press ENTER.

To display current database settings using the command line

1. Click **Start**, click **Run**, type **cmd**, and then click **OK**.
2. At the command prompt, go to the Enterprise Single Sign-On installation directory.

   The default is <drive>:\Program Files\Common Files\Enterprise Single Sign-On.
3. Type **ssomanage -displaydb** and press ENTER.

See Also

Other Resources

Password Synchronization

SSO Security Recommendations

This section contains recommendations for how to help secure your Enterprise Single Sign-On (SSO) system.

With the Enterprise Single Sign-On (SSO) system, users can connect to different systems by using only one set of credentials. Host Integration Server uses the SSO system as a store for sensitive information. Although it automatically installs whenever you install the Host Integration Server runtime, you can also install Enterprise Single Sign-On as a stand-alone component, independent of your Host Integration Server environment. It is recommended you follow these guidelines for securing and deploying the Enterprise SSO services and resources in your environment.

General Deployment Recommendations for SSO

- You must have a time server in your environment to ensure all SSO servers are synchronized. If the clocks on the SSO servers are not synchronized, this could compromise the security of your environment.

- Considering there is only one master secret server in your entire environment, it is recommended that you use an active-passive cluster configuration for the master secret server. For more information about clustering the master secret server, see How to Cluster the Master Secret Server.

- The master secret server holds the encryption key the SSO system uses to encrypt the information in the SSO database. It is recommended that you do not install or configure any other products or services on this computer.

Note
The computer where you install and configure the master secret server does not have to be a server.

- The master secret server should have access to a removable media or NTFS file system folder in order to back up and restore the master secret. If you use removable media, ensure that you take appropriate measures to protect the removable media. If you back up the master secret to an NTFS file system, ensure that you protect the file and the folder. Only the SSO Administrator should have access to this file.

- You should back up the master secret as soon as the master secret server generates it. This is so that you can recover the data in the SSO database in the event the master secret server fails. For more information about backing up the master secret, see Managing the Master Secret.

- Back up your current secret, or generate a new secret regularly, for example, once a month. Without the secret, you cannot retrieve information from the SSO database. For more information about backing up and restoring the master secret, see Managing the Master Secret.

Security Recommendations for SSO Groups and Accounts

- It is recommended that you use Windows groups, and not single user accounts, especially for the SSO Administrator and SSO Affiliate Administrator groups. These groups must have at least two user accounts as members of the group at all times.

- The SSO runtime service accounts and the SSO administrator user accounts should be different accounts, even when they are members of the same SSO Administrators group. The SSO Administrator users who perform administrative tasks such as generating and backing up the secret must be Windows administrators, whereas the SSO runtime service accounts do not need to be Windows administrators.

Security Note
Windows administrator user rights do not supersede the user rights of the SSO administrator. To perform any SSO ad ministration-level task, you must be a member of SSO Administrators group even if you already are a Windows admini strator.

- If you use the SSO ticketing feature, you must use domain accounts that the computers in the processing domain (domain where the SSO servers are) recognize.

- It is recommended that you use a unique service account for the SSO service corresponding to the master secret server.

- The SSO Administrator account is a highly privileged account in the SSO system, which is also the SQL Server administrator account for the SQL server that has the SSO database. You should have dedicated accounts for SSO administrators, and should not use these accounts for any other purposes. You should limit the membership to the SSO Administrators group only to those accounts responsible for running and maintaining the SSO system.

Security Recommendations for an SSO Deployment

- If your network supports Kerberos authentication, you should register all SSO servers. When you use Kerberos authentication between the master secret server and the SSO database, you must configure Service Principal Names (SPN) on the SQL server where the SSO database is located. For more information about configuring Service Principal Names, see the Microsoft Download Web site at http://go.microsoft.com/fwlink/?LinkId=20797.

- When you are running Windows Server 2003, if the master secret server is on a different domain from the other SSO servers and from the SSO database, you must disable RPC security (as used for Data Transaction Coordinator (DTC) authentication between computers) on the master secret server, on the SSO servers (processing computers in the processing domain), and on the SSO database. RPC security is a new DTC feature in Windows Server 2003. When you disable RPC security, the DTC authentication security level for RPC calls goes back to one available in Microsoft Windows 2000 Server. For more information about disabling RPC security, see the Microsoft Help and Support Web site at http://go.microsoft.com/fwlink/?LinkId=24774.

- SSO administrators should regularly monitor the event log in the master secret server and the SSO server for SSO auditing events.

- In addition to firewalls, it is recommended that you use Internet Protocol security (IPsec) or Secure Sockets Layer (SSL) between all the SSO servers and the SSO database. For more information about SSL, see the Microsoft Help and Support Web site at http://go.microsoft.com/fwlink/?LinkId=16731.

Perimeter Network

When running Internet Information Services (IIS) and Enterprise Single Sign-On, follow these recommendations:

- If IIS is in a perimeter network (also known as demilitarized zone, DMZ, and screened subnet), provide another server running IIS behind the firewall to connect to the SSO system.

- Do not open the remote procedure calls (RPC) port on IIS.

SQL Server Access

All SSO servers access the SQL Server Credential database. For more information about how to help secure SQL Server databases, see http://go.microsoft.com/fwlink/?LinkId=33175.

It is recommended that you use Secure Sockets Layer (SSL) and/or Internet Protocol security (IPsec) to help secure the transmission of data between the SSO servers and the Credential database. For more information about using SSL, see http://go.microsoft.com/fwlink/?LinkId=33176.

To enable SSL for only the connection between the SSO server and the Credential database, you can set SSL support on every SSO server using the ssoconfig utility. This option enables SSO to always use SSL when accessing the Credential database. For more information, see How to Enable SSL for Enterprise Single Sign-On.

Strong Passwords

It is very important that you use strong passwords for all accounts, especially the accounts that are members of the SSO Administrators group, because these users have control over the entire SSO system.

SSO Administrator Accounts

It is recommended that you use different service accounts for the SSO services running on different computers. You should not use the SSO administrator account that performs administration operations such as generating and backing up the secret for

the SSO service. Although the SSO service accounts should not be local administrators on that computer, the SSO administrator who is performing administration operations must be a local administrator on the computer for some operations.

Master Secret Server

It is highly recommended that you secure and lock down the master secret server. You should not use this server as a processing server. The only purpose of this server should be to hold the master secret. You should ensure the physical security of this computer and only SSO Administrators should have access to this computer.

Kerberos

SSO supports Kerberos, and it is recommended that you set up Kerberos for SSO. To set up Kerberos with SSO, you must register a Secure Principal Name (SPN) for the SSO service. By default, when you set up Kerberos, SSO uses that SPN to authenticate the components using the SSO Service. It is recommended you set up Kerberos authentication between the SSO administrative sub services and the SSO server. You can also user Kerberos authentication between the SSO servers and between the SSO servers and the SQL Server where the Credential database is.

To set up and verify Kerberos, you use the utilities **setspn** and **kerbtray**. For more information about these utilities, see http://go.microsoft.com/fwlink/?LinkId=33178 and http://go.microsoft.com/fwlink/?LinkId=33179.

Delegation

When you are using Windows Server 2003, you can use constrained delegation, but it is recommended that you do not use delegation to perform the tasks of the Single Sign-On administrator. Similarly, it is recommended that you do not delegate additional tasks or user rights to the Single Sign-On administrator.

Auditing

Auditing is a critical mechanism for tracking information in your environment. Enterprise Single Sign-On (SSO) audits all operations performed in the Credential database. SSO uses event logs and audit logs of the database itself. SSO provides two audit levels for the Single Sign-On servers:

- Positive auditing levels audit successful operations.
- Negative auditing levels audit operations that fail.

SSO administrators can set the positive and negative audit levels that suit their corporate policies.

You can set positive and negative audits to one of the following levels:

- 0 = None - This level issues no audit messages.
- 1 = Low
- 2 = Medium
- 3 = High - This level issues as many audit messages as possible.

The default value for positive auditing is 0 (none), and the default value for negative auditing is 1(low). You may want to change these values depending on the level of auditing you want for your SSO system.

Important
Enterprise Single Sign-On auditing issues messages that are generated by the Single Sign-On service. This is not a security a udit, and the SSO system does not save the information in the Security log of the Event Log. The SSO system saves the SSO a udit messages directly to the Application Event Log.

Database-Level Auditing

For database-level auditing, the SSO system tracks the operations performed on the Credential database in the audit tables in the database. The size of these audit tables are defined at the SSO system level. You can audit for affiliate applications that are deleted, for mappings that are deleted, and for credential look-ups that are performed. By default, the audit size is set to 1000 entries. SSO administrators can change this size to meet their corporate policies.

Using Enterprise Single Sign-On Accounts

This section contains best practices when you are using domain and local groups and individual accounts in the Enterprise Single Sign-On (SSO) system.

Domain Windows Groups and Accounts

When you are working with domain Windows groups, the following recommendations apply:

- Use domain groups and domain accounts.
- Use a domain group for SSO administrators. You should not specify an individual domain account as the SSO administrator, because you cannot change this account from one individual account to another individual account.
- Although you can specify an individual domain account as the SSO affiliate administrator, you should use a domain group.
- Although you can specify an individual domain account as the application administrator, you should use a domain group.
- You must use domain groups for the application users account. The SSO applications users account does not support an individual account.

See Also

Tasks

How to Audit Enterprise Single Sign-On

How to Update the Credential Database

Transaction Integrator User's Guide

This section contains information about using Transaction Integrator (TI). Transaction Integrator is the synchronous COM+ or .NET Framework application integration solution in Host Integration Server. TI enables you to integrate mainframe-based transaction programs (TP) and AS/400 transactions with component-based Windows Server System applications when the following conditions are true:

- A synchronous or transactional solution is needed.
- Both the client and server systems are running at the time the call is made.

If you need an application integration solution that does not require the client and server systems to be running at the time the call is made, use an asynchronous messaging solution such as the MSMQ-MQSeries Bridge instead of TI. In an asynchronous solution, the middle-tier queuing system is running at the time the client issues a request message, the server retrieves the message and sends back the reply, and then the client receives the reply back from the middle tier.

With TI, you can integrate existing mainframe-based TPs with Windows-based COM, distributed COM (DCOM), or applications built on the .NET Framework. You might not even have to modify your TP if you have separated the business logic from the presentation logic. The wizards in TI guide you through the modification process, step by step.

With TI, you can preserve existing CICS and IMS TPs as you move to a three-tier client/server or Web-to-host computing environment. By using TI to invoke mainframe transactions, you can program in the visual object-oriented environments and programming languages that you know while you maintain access to host transactions.

TI supports both SNA connectivity and TCP/IP connectivity without requiring a host footprint or costly host transaction rewrites. You can choose SNA connectivity if you need two-phase commit (2PC), or choose TCP/IP connectivity if you need direct throughput. IBM has not implemented 2PC for the TCP/IP protocol, but for those cases where 2PC is not necessary, TCP/IP can give you direct connectivity.

True integration of online transaction processing (OLTP) with COM- or .NET-compliant systems means the integration of CICS and IMS with Windows-based solutions. CICS and IMS are widely used in the mainframe arena to create distributed OLTP solutions such as customer tracking and order entry. TI integrates CICS and IMS with COM by creating COM interfaces or .NET interfaces to the CICS and IMS transactions and then running the CICS and IMS transactions on the mainframe from Windows.

A TI component in a COM+ application works in concert with the TI run-time environment, Microsoft Distributed Transaction Coordinator (MS DTC), and the associated remote environment (RE) to drive a CICS or IMS TP. Together, they accomplish these tasks:

- Activate the host (mainframe) TP.
- Pass the parameters specified by the TI component to the TP.
- Run the TP.
- Return the results to the TI component.

When you deploy a TI component (a type library .tlb file) in a COM+ application, that COM+ application becomes a TI *Automation server*. When a client application invokes a method in that TI Automation server, Windows automatically starts the TI run-time environment in the associated remote environment to invoke the mainframe transaction that is associated with that TI method. Component Services in Windows 2000 automatically handles any class factory, early or late binding, or other internal operations needed. The invoked mainframe transaction can call other transactions on the mainframe before it returns the result to the COM-based client application through the TI Automation server.

In This Section

Getting Started with TI

This section explains some of the basics of using Transaction Integrator.

In This Section

Remotely Administering Transaction Integrator

Transaction Integrator (TI) does not support remote installation or configuration, and it does not support remote administration of Windows-initiated processing (WIP) except through use of the Windows Remote Desktop. TI does support remote administration of host-initiated processing through the use of TI Manager and the Remote Desktop. TI also supports the use of remote SQL Server databases.

Remote Installation and Configuration

TI must be installed and configured locally, and it does not support remote installation or configuration. The only way to install TI Manager and the TI runtime is by using the **Host Integration Server Installation Wizard**, and selecting **Application Integration** on the options page. You cannot install TI Manager or the TI runtime programmatically.

Remote Administration of Windows-Initiated Processing

You can administer Windows-initiated processing only from the local computer, and TI does not support remote administration for WIP. Each computer that has TI installed and configured and that is used for WIP is administratively separate from any other computer that also has TI installed. If you need to administer WIP when you are away from the local computer, you must use the Windows Remote Desktop feature to access TI Manager and the TI runtime.

Remote Administration of Host-Initiated Processing

You can administer the host-initiated processing remotely, but only from a computer that also has TI installed on it and uses the same HIP configuration database. For example, if three servers each have TI installed on them, and all three are configured to use the same HIP database, then you can remotely administer any of the three servers from either of the other two servers. You cannot remotely administer the TI runtime from another server that has TI installed but uses a different HIP configuration database. For example, if you have a fourth server that also has TI installed but uses a different HIP configuration database from the three servers in the previous example, you cannot remotely administer the TI runtime on any of the three other servers. Neither can you remotely administer the fourth server from any of the other three servers.

If you need to administer HIP when you are away from a computer that is configured to use the same HIP configuration database, you must use the Windows Remote Desktop feature to access TI Manager and the TI runtime.

Use of Remote Databases

You can also configure TI to use a remote SQL Server database. The SQL Server database is used to store TI host-initiated processing (HIP) configuration data. Use the **Microsoft Host Integration Server Configuration Wizard** to specify the database server to use.

To configure a remote database

1. Click **Start**, point to **Programs**, point to **Microsoft Host Integration Server**, and then click **Configuration Wizard**.
2. Follow the directions on the screen.
3. On the **Database configurations** wizard page, select **Transaction Integrator Configuration Database**.
4. Click **Edit**.
5. In the **Transaction Integrator Configuration Database Properties** dialog box, select the remote server and database to be used.
6. Click **OK**, and then click **Next**.
7. Continue to follow the directions on the screen.

After you click **Finish** in the Wizard, TI installs the HIP database on the designated remote server.

See Also
Concepts
Getting Started with TI
Other Resources
Using Windows-Initiated Processing
Using Host-Initiated Processing

Transaction Integrator Manager Console

Transaction Integrator is administered through the Transaction Integrator (TI) Manager management console. Microsoft Management Console (MMC) is a framework for hosting administrative tools called *snap-ins*. In general, a console can contain tools, folders or other containers, Web pages, and other administrative items. These items are displayed in the left pane of the console, or the *console tree*. MMC provides a single user interface for integrating various Microsoft and third-party management tools. You can create custom consoles combining your choice of the various registered snap-ins; for example, you might put the SNA Manager and Transaction Integrator snap-ins together in a single console.

The TI Manager management console consists of a left pane that displays the console tree and a right pane that displays the properties of the objects in the left pane. The topmost node in the console tree is the **Console Root**, and as you add snap-ins to MMC, they appear under the **Console Root**.

Console Root Nodes

When you first open TI Manager, the console root has three major nodes (or snap-ins):

- **Transaction Integrator**. This node is the entry point to the TI Manager administrative tool.

- **Component Services**. This node is the entry point to the Component Services administrative tool. The Component Services administrative tool enables you to configure and administer COM components and COM+ applications, including installing and configuring COM+ applications, setting security at the application level, and creating and maintaining COM+ partitions. The Component Services administrative tool is designed both for system administrators and for developers. For example, developers can configure routine component and application behavior, such as participation in transactions and object pooling.

- **Internet Information Services**. This node appears in the console only if Internet Information Services (IIS) is installed on the computer. The **Internet Information Services** node is the entry point to the IIS administrative tool.

Transaction Integrator Node

The **Transaction Integrator** node contains two major subnodes:

- **Host-Initiated Processing**. You can use this node to view the major elements that are used in a host-initiated processing (HIP) environment.

- **Windows-Initiated Processing**. You can use this node to view the major elements that are used in a Windows-initiated processing (WIP) environment.

See Also

Reference

Transaction Integrator (mode) Node
Host-Initiated Processing Node
Windows-Initiated Processing Node

Other Resources

Transaction Integrator Manager Help

Using Windows-Initiated Processing

The following topics explain how to use Windows-initiated processing (WIP) over TCP/IP and over SNA.

In This Section

Where to Begin

Transaction Integrator (TI) enables you to integrate mainframe-based transaction programs (TP) with COM-based client applications by creating and using a TI Automation server to invoke a TP on a mainframe from a COM-based client application.

To learn TI and to design and plan your first TI system, complete the following tasks:

- **Learn the basics**

  Learn about each of the elements in a TI system and how they interoperate. You can use this understanding to determine whether you need to make any changes to your mainframe-based TP or to your COM-based client application. The TI Automation server (a TI component deployed in a COM+ application) is the software that forms the bridge between the mainframe transactions and the client application. For more information, see Learning the Basics.

- **Install TI**

  Install TI when you install Host Integration Server Server on the computers that you will use to create, deploy, and manage TI components.

- **Configure an SNA or TCP/IP connection**

  Configure an SNA connection from your computer to the mainframe by using Host Integration Server, or configure a TCP/IP connection. Even though Host Integration Server is not required to configure a TCP/IP connection, you must have Host Integration Server to install TI.

- **Create LUs and an APPC Mode for an SNA connection**

  Create local and remote logical units (LU) and an Advanced Program-to-Program Communications (APPC) Mode for all SNA connections.

Note
If you use .NET to access the remote server, the .NET Auto Web Proxy feature may cause a significant reduction in throughput. To disable this feature and achieve maximum throughput, add the following line to your WIP client code: System.Net.WebRequest.DefaultWebProxy = null;

To start using TI to connect COM-based applications with mainframe-based TPs, complete the following general steps:

To start using TI to connect COM-based applications with mainframe-based TPs

1. In TI Manager, create an appropriate Customer Information Control System (CICS) or Information Management Systems (IMS) remote environment (RE) and configure its properties.

   For help, see Creating a Remote Environment.

2. In Host Integration Server Designer, create a TI component for a specific RE.

   For help, see Creating TI Components.

3. Deploy the TI component by adding it to a COM+ application.

   This process creates the TI Automation server that any COM-based application can use to invoke the TP. For help, see How to Deploy a TI Component.

4. Test the new TI automation server by calling its methods from the Automation client application.

   This action will run the appropriate transactions in the TP on the mainframe. For help, see Testing a TI Automation Server.

5. Put the new, fully-tested TI Automation server into production; use TI Manager to reconfigure and manage the TI Automation server, its associated RE, and the TI run-time environment settings as needed.

See Also

Other Resources

Using Windows-Initiated Processing

Learning the Basics

Before you implement Transaction Integrator (TI), scan the topics in this section to gain a basic understanding of TI. This information is necessary to properly plan your TI implementation and make any necessary modifications.

TI System Elements

A TI system includes the following elements:

- A region on a mainframe computer and the COBOL transaction programs (TP) that run in that region. You do not need knowledge of COBOL, but you do need to understand what each of the component transactions in a mainframe-based TP does, as well as how TPs interact with each other and the total system. Each transaction in a Customer Information Control System (CICS) or Information Management Systems (IMS) TP on the mainframe can be invoked by a method defined in a TI component. Because after a mainframe transaction in a TP is invoked, it can call transactions in other TPs, it is important to know precisely what each transaction in the TP does. To be precise, each method in a TI Automation interface calls a single mainframe TP. It is the TP that determines which mainframe transaction to call. The mainframe TP makes this determination based on the information passed to it by the TI method.

- Component Object Model (COM) distributed COM (DCOM), and Component Services (COM+) in Windows 2000. For information about these elements, see Introduction to COM and COM+.

- CICS or IMS, remote and local independent logical units (LU), and APPC modes associated with the mainframe. See IBM documentation for information about these elements.

- Windows 2000 or later services, including the System Monitor and Event Viewer. Refer to the Windows Help for information about the Windows operating system.

- Host Integration Server server with which you connect your Windows computer to the mainframe. Review the Help file for this information.

- SNA or TCP/IP protocols. These are the IBM protocols that are supported by TI. See IBM documentation for this information.

- A remote environment (RE) associated with each TI component to define the mainframe environment that the TI component will use. To create and modify REs, use TI Manager.

- Transaction Integrator, which includes three tools: Host Integration Server Designer (HIS Designer), TI Manager, and the TI run-time environment. Both HIS Designer and TI Manager have context-sensitive Help that you can consult.

- A COM-based client application that calls the methods of a TI Automation server (a TI component deployed in a COM+ application) to interoperate with mainframe TPs.

- A TI component. This component wraps COM around the TP's data definition section. To create a new TI component or modify an existing one, use HIS Designer. To deploy a TI component in a COM+ application, use TI Manager.

- A COM+ application in which a TI component is deployed. After the TI component is deployed in a COM+ application, that COM+ application becomes a TI Automation server. The COM+ application provides the Automation interface for your TI Automation server. A client application can use the services of the new TI Automation server by way of the Automation interface.

- Microsoft Distributed Transaction Coordinator (DTC) included in and used internally by Component Services (COM+). You have to know that Microsoft DTC is the actual transaction manager, but you do not have to understand how it works.

Additional Information Sources

In addition to exploring the TI documentation, you can obtain other useful information, such as the Win32 Platform SDK documentation, from these Microsoft Web sites:

- The MSDN Online Library at http://go.microsoft.com/fwlink/?LinkId=12768 provides the Platform SDK, other SDK documentation, DDK documentation, Windows resource kits, technical programming information, sample code, technical articles, backgrounders, specifications, and reference guides.

- The MSDN Online home page at http://go.microsoft.com/fwlink/?LinkId=12768.

- The MSDN Online Web Workshop at http://go.microsoft.com/fwlink/?LinkId=12768 provides the latest information about Internet technologies, including reference material and in-depth articles on all aspects of Web site design and development.

- The Microsoft COM site at http://go.microsoft.com/fwlink/?LinkId=12800 gives you the latest information about Microsoft Component Object Model (COM), distributed COM (DCOM), and other COM-based technologies. You will find white papers, presentations, case studies, files to download, samples, the COM and DCOM specifications, a list of training courses, a list of other helpful sites, and a list of useful books. The covered technologies include COM, DCOM, COM+, and ActiveX.

See Also
Tasks
Where to Begin

How to Create a Remote Environment

A remote environment (RE) holds the information about a particular mainframe region that the Transaction Integrator (TI) run-time environment needs to interact effectively with the mainframe environment. Use TI Manager to create the remote environments that you need.

The following are the RE types that you can create in TI Manager:

- Customer Information Control System (CICS) and Information Management System (IMS) using TCP/IP
- CICS LINK, using LU 6.2
- CICS (non-LINK), using LU 6.2
- Diagnostic Capture
- Diagnostic Playback
- IMS using LU 6.2
- IMS using IMS Connect or OTMA over TCP/IP

Note that there is no support for 3270-oriented transaction programs (TP).

You must associate every TI component that you create in Host Integration Server Designer (HIS Designer) with an RE of one of the following types (classes):

- CICS and IMS using TCP/IP
- CICS LINK using LU 6.2
- CICS (non-LINK) using LU 6.2
- IMS using LU 6.2
- IMS using IMS Connect or OTMA for TCP/IP

Once you create an instance of the RE, you can associate one or more TI components with that RE.

After you create the RE in TI Manager and create the TI component in HIS Designer, you are ready to deploy the TI component in a COM+ application by dragging the file from Windows Explorer and dropping it into a COM+ application in TI Manager. When you deploy a TI component, TI Manager automatically assigns that TI component to the default RE for the RE type specified in the TI component. If there is no default RE, TI Manager places the component in the Unassigned Components folder. For more information see How to Deploy a TI Component.

Before creating a new RE, you will need some information from the network administrator.

- To create a Diagnostic Capture RE or a Diagnostic Playback RE, you need to know which of the following RE types your TI component is designed to use:
    - CICS and IMS using TCP/IP
    - CICS LINK using LU 6.2
    - CICS (non-LINK) using LU 6.2
    - IMS using LU 6.2

- IMS using IMS Connect or OTMA for TCP/IP

- In the case of the Diagnostic Playback RE, you also need to know the name and location of the .rcd file that contains the recording.

- To create any of the TCP/IP-based REs, you need to know the IP address and the TCP ports list number of the IBM Listener for the CICS or IMS region in which the transaction program is deployed.

- To create any of the SNA (APPC/LU 6.2)-based REs, you need to know the local LU alias, the remote LU alias, and the APPC mode name.

To create a new RE for a TI component built in HIS Designer

1. Start TI Manager.
2. Expand the **Transaction Integrator** folder.
3. Right-click the **Remote Environments** folder, point to **New**, and then click **Remote Environment**.
4. Click one of the following RE types, and then click **Next**.
    a. CICS and IMS using TCP/IP

    b. CICS LINK using LU 6.2

    c. CICS (non-LINK) using LU 6.2

    d. IMS using LU 6.2

    e. IMS using IMS Connect or OTMA for TCP/IP

5. Enter the requested information in the pages that follow, and then click **Finish**.

See Also
Concepts
Creating a Remote Environment
Other Resources
Using Windows-Initiated Processing

Creating TI Components

Although you can create and test a Transaction Integrator (TI) component even before you have a connection to the mainframe, it is preferable to create and test your TI component after you have established a connection.

Use Host Integration Server Designer (HIS Designer) to build the TI component. While you are building the component, you are asked to associate it with an appropriate remote environment (RE). The wizards in HIS Designer guide you through the process, step-by-step. In the Object Creation Wizard, you are asked to select whether the component you are creating is Windows initiated or host initiated. As shown in the following graphic, select Windows Initiated.

Use HIS Designer to define a TI component to use a specific kind of environment for each TI component that you create.

Supported Programming Models

HIS Designer supports the following TCP/IP and SNA (APPC/LU 6.2) mainframe programming models:

- CICS Concurrent Server (TCP/IP)
- CICS MS Link (TCP/IP)
- IMS Implicit (TCP/IP)
- IMS Explicit (TCP/IP)
- IMS using IMS Connect or OTMA (TCP/IP)
- CICS LINK (APPC/LU 6.2)
- CICS non-LINK (APPC/LU 6.2)
- IMS (APPC/LU 6.2)

In HIS Designer, you can create TI components (type library .tlb files) that use each of these environments. The first four types listed are all associated with a CICS and IMS using TCP/IP RE that you created in TI Manager. The more detailed definition is needed in HIS Designer so that the data buffer can be laid out properly, and so that various checks that HIS Designer makes can be completed successfully.

In addition, in HIS Designer, you can specify whether or not a CICS LINK RE should allow 32 KB in each direction (in and out). If you do not specify each direction, 32 KB is the maximum for the sum of both directions (in and out); for example, you could have 16 KB in and 16 KB out, or 1 KB in and 31 KB out. This applies only when you are creating a component; you do not need to specify this when you are defining an RE in TI Manager. It matters for the TI component definition for two reasons:

- It affects how the data is laid out in the buffer.
- It affects how much data you can have in the method.

By configuring a TI component to use one of the eight supported environments, you can choose to communicate directly over TCP/IP or use LU 6.2, an SNA protocol. TI supports two-phase commit (2PC) for ACID (atomic, consistent, isolated, and durable) distributed transaction processing only over SNA LU 6.2networks. On the mainframe, 2PC is done with Sync Level 2 support. IBM does not yet support 2PC over the TCP/IP protocol, except for CORBA using the IIOP protocol. Therefore, for those TPs where you need to maintain distributed transactional integrity, use the APPC/LU 6.2 protocol.

See Also
Concepts
Creating and Managing TI Components
Other Resources
Using Windows-Initiated Processing

How to Deploy a TI Component

To deploy a Transaction Integrator (TI) component, add it to a COM+ application. A TI component consists of a type library (.tlb file) that is created with Host Integration Server Designer (HIS Designer). Typically, you will deploy all TI components that you need for your Automation server application in a single COM+ application. You do not have to deploy other files, such as associated DLLs, together with a TI component library. When you add the TI component library in a COM+ application, that COM+ application becomes an Automation server that is automatically associated with the generic TI run-time environment code (tagen.dll).

After you deploy a TI component, you can view the component's interfaces and methods. To check the deployed component's property settings to verify that it is associated with the correct remote environment, right-click the component name, and then click **Properties**.

Use any of the following three methods to deploy a TI component. To use these procedures, you must have administrator privileges.

To deploy a TI component from HIS Designer

1. In HIS Designer, click the interface for the component you want to deploy.
2. On the **Tools** menu, click **Add To Package or Application**.
3. Click the appropriate package or application.
4. Click the appropriate remote environment for the TI component that is being deployed, and then click **OK**.

To deploy a TI component from TI Manager

1. Start TI Manager.
2. In the console tree, double-click the **Component Services** (or Microsoft Transaction Server) folder, double-click the **Computers** folder, double-click the folder for the computer in which you want to deploy the component, and then double-click that computer's **COM+ Applications** folder.
3. In the console tree, double-click the specific COM+ application in which you want to deploy the TI component.

   If you have not yet created a COM+ application for the component, right-click the **COM+ Applications** folder in the console tree, point to **New**, and then click **Application** to start the wizard. Follow the instructions to create the new COM+ application.
4. In the console tree, click the **Components** folder under the folder for the target COM+ application.
5. On the **Action** menu, point to **New**, and then click **Component** to start the wizard.

   (If you are using Windows 2000, the wizard asks you if you want to install a component, import a component that is already registered, or install a new event class; click the button next to **Install new component(s)**. You cannot import a TI component; you must install it (that is, deploy it). Finish deploying your TI component (.tlb file) by specifying its path and file name.)

To deploy a TI component by dragging it from Windows Explorer

1. Start TI Manager.
2. In the console tree, double-click the **Component Services** (or Microsoft Transaction Server) folder, double-click the **Computers** folder, double-click the folder for the computer where you want to deploy the component, and then double-click that computer's **COM+ Applications** folder.
3. In the console tree, double-click the COM+ application in which you want to deploy the TI component.

   If you have not yet created a COM+ application for the component, click the **COM+ Applications** folder in the console tree. Then right-click that folder, point to **New**, and click **Application** to start the wizard. Follow the instructions to create the new COM+ application.
4. In the console tree, click the **Components** folder for the target COM+ application.

   At this point, you will see the contents of the folder (if any) displayed in the details pane.
5. Resize or reposition the TI Manager window so that you have room on the desktop to open and use Windows Explorer.

6. Open Windows Explorer and locate the folder that contains the TI component (.tlb file) that you want to deploy.
7. Drag the .tlb file from Windows Explorer, and drop it into the details pane of TI Manager.

See Also

Tasks

Where to Begin

Concepts

Creating and Managing TI Components

Testing a TI Automation Server

You can test your Transaction Integrator (TI) Automation server by calling the methods on its Automation interface from a COM-based client application.

See Also
Tasks
Where to Begin

Creating a Remote Environment

Remote environment (RE) definitions are created and managed from the Remote Environments folder. When the WIP (Windows-initiated processing) Console is first started, the remote environment folder is empty.

A remote environment defines the network, hardware, and software characteristics of the non-Windows host that will be receiving requests from the Windows operating system. For example, a CICS host that uses the TCP/IP protocol to receive requests from Windows is identified to WIP through an RE that contains the IP address of the Host, the Port Number being listened on, and the code page that the host uses to represent its data.

In This Section

Windows-Initiated Processing Console

Starting the New Remote Environment Wizard

How to View All Remote Environments

Adding a Remote Environment

See Also

Other Resources

Using Windows-Initiated Processing

Windows-Initiated Processing Console

The Windows-initiated processing (WIP) console supports the COM, .NET Framework, TCP/IP, and SNA environments, with COM objects and .NET Framework assemblies having similar capabilities and functioning.

There are two primary configuration elements for the WIP environment:

- **Remote environments.** The remote environments folder contains the definitions, called remote environments or REs, for the non-Windows host computers that receive requests from the WIP components.

- **Objects.** The objects folder contains the metadata definitions for the client proxy objects that were created through the Host Integration Server Designer (HIS Designer).

The Component Services node is used to configure COM+ packages for WIP COM objects. The Internet Information Services node is used to configure the virtual directories used by WIP .NET Framework objects.

In This Section

Remote Environments

Objects

Relationships

See Also

Concepts

Creating a Remote Environment

Creating and Managing Remote Environments Using TI Manager

Remote Environments

The Remote Environments folder contains definitions for the non-Windows host computers that will receive requests from the Windows-initiated processing (WIP) components. These host definitions are referred to as remote environments or REs.

The RE is used by the WIP Runtime for the following primary purposes:

- Define the code page used by the host.
- Define the data conversion object that will be used by the WIP Runtime.

Properties on each RE define the characteristics of the host that will be receiving requests.

REs Supported by WIP

WIP supports multiple types of REs (for backward compatibility reasons, there are not just two, as in host-initiated processing).

The following is a minimum set of REs that is supported:

- CICS and IMS using TCP/IP
- CICS LINK using LU6.2
- CICS using LU6.2
- IMS Connect
- IMS using LU6.2

Each of the previous RE types has a (possibly) unique collection of properties, some of which are in conjunction with the Host Security functionality.

TCP/IP RE Properties

The basic TCP/IP RE properties contain the following:

- IP Address or Host Name
- Port List
- Receive timeout value
- Locale and Code Page
- Windows Security Context to be used (User or Package)

SNA RE Properties

The basic SNA RE properties contain the following:

- Local and Remote LU Aliases, Mode Name
- Receive timeout value
- Support for Sync Level 2
- Locale and Code Page
- Windows Security Context to be used (User or Package)

See Also

Tasks

How to View All Remote Environments

Concepts

Windows-Initiated Processing Console

Creating a Remote Environment

Objects

The Objects folder contains the metadata definitions for the Client Proxy objects that were created through the Application Integrator Designer. These metadata definitions contain the following:

- The COM ProgID of the callable COM object (in the form of a type library).
- The .NET Framework qualified namespace of the callable .NET Framework class (in the form of a .NET Framework assembly containing a type library)
- Conversion annotations used by the Windows-initiated processing (WIP) Runtime to interoperate with the host application program environment.
- A representation of the callable COM or .NET Framework object

The metadata definition files are Transaction Integrator metadata files (*.tlb or *.dll) generated by the Application Integrator Designer. The folder contains information about finding these metadata files and their relationships with remote environments and COM+ Packages (COM) or IIS Virtual Directories (.NET).

See Also
Concepts
Windows-Initiated Processing Console

Relationships

A configured Windows-initiated processing (WIP) environment is established by defining a set of WIP elements and then relating them in a manner that enables the client/client program, COM+ / .NET Framework remoting and the WIP Runtime to do the following:

- Write a program against a COM (type library) or .NET proxy (assembly).
- Locate the Proxy (in a COM+ Package or IIS Virtual Directory).
- Make a client call.
- Transform the incoming parameters from COM or .NET formats to wire format data streams.
- Select the correct Transport protocols and destination for the call.
- Transfer data to and from the host application
- Transform the data stream to a COM or .NET reply (parameters, return values, in the format expected, for example, **RecordSets** vs. **DataTables**)
- Return to the calling application.

The Object is the entity that the client "sees" as executed. Methods on the object are executed on a particular host based on any Selection Hint given or their associated remote environment (RE). There is a one-to-many relationship between an RE and an Object that allows for multiple Objects to be executed via the same Host definitions given by an RE.

The WIP Microsoft Management Console (MMC) Administrative Console enables these relationships to be established by using wizards or through manual configuration. After these relationships are established, client calls that result in host processing can be initiated.

See Also
Concepts
Windows-Initiated Processing Console

Starting the New Remote Environment Wizard

You can use the **New Remote Environment Wizard** to define the characteristics of the environment that receives request from the Windows system.

The remote environment definition includes:

- Remote Environment Name
- Network Transport type
- Host identification
- Host Software Environment
- Code Page

See Also
Concepts
Windows-Initiated Processing Console

How to View All Remote Environments

Follow these steps to view all remote environments (RE):

To view the names and properties of the remote environments

1. In the left pane, click the **Remote environments** folder, and then click an RE name.
2. In the right pane, view the name and properties of each application running on the computer.
    - **Remote Environment**
    - **Type**

To view the operations available on a remote environment

1. In the left pane, click the remote environment name.
2. In the right pane, click the desired action:
    - **New**
    - **View** displays a list of menu items:

      Add/Remove Columns enables you to choose which properties are displayed in the list view. All properties of the remote environment are viewable as a column.

      Large Icons displays larger icons for items in the right pane.

      Small Icons displays smaller icons for items in the right pane.

      List displays only the smaller icons and the names of the applications in the right pane.

      Detail displays the smaller icons and the properties of the applications in the right pane.

      Customize enables you to change the options to show or hide items displayed in the right pane.
    - **Delete** deletes the remote environment from the computer. The deleted item is removed from the specific computer that it was defined on and from the administrative data store.
    - **Rename** renames the highlighted remote environment. The new name must be unique and is reflected across all elements of the host-initiated processing (HIP) console. The **Rename** operation does not affect the operation of Objects that have references to the remote environment. After the **Rename** operation is completed, any Objects that reference the remote environment continue to maintain their reference to the renamed remote environment.
    - **Refresh** redraws the screen to show any updates.
    - **Export List** enables you to save the list of remote environments as a separate file.
    - **Properties** enables you to view and change the properties of a remote environment.
    - **Help** displays an online Help topic explaining the items that appear in the Transaction Integrator (TI) Manager console and the actions you can take.

See Also
Concepts
Creating a Remote Environment

Adding a Remote Environment

You can use the **Remote Environment Wizard** to define the characteristics of the environment that receives request from the Windows system.

The remote environment definition includes the following:

- Remote Environment Name
- Network Transport type
- Host identification
- Host Software Environment
- Code Page

See Also
Concepts
Windows-Initiated Processing Console

Creating an Object

Objects in the Windows-initiated processing (WIP) environment represent the “Servers” that can be called by a client. As a result of this call, a request is sent to a host. The Object is defined by a Transaction Integrator Metadata file that is generated by the development-time tool (Application Integrator Designer).

You can use the **Object Wizard** to create new objects.

See Also

Concepts

Objects

Windows-Initiated Processing Console

Other Resources

Using Windows-Initiated Processing

Viewing All Objects

Click **View All** to display the entire list of objects with their corresponding properties.

See Also

Concepts

Creating an Object

Objects

Other Resources

Using Windows-Initiated Processing

Creating and Managing Remote Environments Using TI Manager

Use Transaction Integrator (TI) Manager to create and manage remote environments (RE) for each mainframe region.

In This Section

How to Start TI Manager

You can start TI (Transaction Integrator) Manager many different ways:

- Use the **Start** menu
- Start Microsoft Management Console (MMC), and then add the appropriate snap-in consoles
- Double-click a saved MMC configuration file (.msc file)
- Enter a command line at a command prompt
- Use the **Run** command on the **Start** menu

You can also start TI Manager from MMC. Then you can save your MMC configuration as an .msc file, so that later you can just double-click the .msc file to start it. This technique can be useful when you are adding other snap-ins to MMC in addition to the TI Manager snap-ins.

To start TI Manager from the Start menu

- Click **Start**, point to **Programs**, point to **Host Integration Server**, point to **Application Integration**, and then click **TI Manager**.

To start TI Manager from MMC

1. Start MMC by doing one of the following:
    a. Click **Start**, click **Run**, type **mmc**, and then click **OK**.
    b. Type **mmc** at a command prompt, and then press ENTER.
2. On the **Console** menu, click **Add/Remove Snap-in**.
3. In the **Add/Remove Snap-in** dialog box, click **Add**.
4. In the **Add Standalone Snap-in** dialog box, double-click **TI Manager**.
5. In the **Specify machine to administer** dialog box, click **Local computer**.
6. Click **Finish**.
7. In the **Add Standalone Snap-in** dialog box, double-click **Component Services** (or **Microsoft Transaction Server**).
8. In the **Add Standalone Snap-in** dialog box, click **Close**.
9. In the **Add/Remove Snap-in** dialog box, click **OK**.

    If you save the configuration as, for example, MyCOMTI.msc, you can run it again later by double-clicking the file name.

To start TI Manager from the command

1. Click **Start**, and then click **Run**.
2. Type the following:

    MMC "c:\Program Files\Host Integration Server\System\ComtiComPlus.msc"
3. Click **OK**.

    Make sure that you include the quotation marks because the path contains spaces.

Starting TI Manager from the Command Line

You can also start TI Manager from a command line. Include the path and file name of the .msc file when you use the **Run** command or from a command prompt. Microsoft has included two .msc files (ComtiComPlus.msc and ComtiMTS.msc) for you

in the Host Integration Server\System folder. The following procedure is an example.

If you previously added TI Manager to a remote computer, you can use the command line to start TI Manager for the remote computer if you know the name of that computer. However, you must have enabled the following option at the time that you added the console for the remote computer: Allow the selected computer to be changed when launching from the command line. For example, if you created a file named MyOtherCOMTI.msc on another computer in a Windows 2000 network, you can run it on your local computer by using the following procedure.

To start a remote TI Manager from the command line

1. Click **Start**, and then click **Run**.
2. Type the following:

   MMC "C:\Program Files\Host Integration Server\System\MyOtherCOMTI.msc" /computer=computername
3. Click **OK**.

   Adding the **/computer=computername** option lets you configure and otherwise manage all your TI environments from a single computer.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager

Creating and Managing TI Components

How to Define an SNA CICS or SNA IMS Remote Environment

An SNA CICS or SNA IMS remote environment (RE) is an RE of the CICS Using LU 6.2, CICS LINK Using LU 6.2, or IMS Using LU 6.2 RE type.

To define an SNA CICS or SNA IMS RE

1. Start TI Manager.
2. Double-click **Transaction Integrator** in the console tree.
3. Right-click **Remote Environments**, point to **New**, and then click **Remote Environment**.
4. In the **Add Remote Environment** dialog box, click an RE type (CICS Using LU 6.2, CICS LINK Using LU 6.2, or IMS Using LU 6.2), and then click **OK**.
5. Specify the SNA attributes for the RE, and then click **Next**.
6. Enter a name for the new RE instance, or accept the default.

   The name can be a maximum of 255 characters.
7. Enter an optional comment to describe any distinguishing characteristics of this RE.
8. Click **Next**, and then click **Finish**.

See Also
Concepts
Creating and Managing Remote Environments Using TI Manager
Getting Started with TI

How to Define a Transactional SNA CICS or SNA IMS Remote Environment

A transactional SNA CICS or SNA IMS remote environment (RE) is one that supports Sync Level 2 to enable two-phase commit (2PC). Note that none of the TCP/IP REs support 2PC because TCP/IP does not support Sync Level 2 or any other 2PC mechanism.

Note that IMS version 6.0 and IBM Resource Recovery Service (RRS) are required on the mainframe system for IMS sync level 2 support.

To define a transactional SNA CICS or SNA IMS RE

1. Start TI Manager.
2. Create a new CICS Using LU 6.2, CICS LINK Using LU 6.2, or IMS Using LU 6.2 RE, or click an existing SNA RE in the console tree.

   To create a new RE, double-click **Transaction Integrator** in the console tree, right-click **Remote Environments**, point to **New**, and then click **Remote Environment**.
3. Right-click **CICS Using LU 6.2**, **CICS LINK Using LU 6.2**, or **IMS Using LU 6.2 RE**, and then click **Properties**.
4. Click the **LU 6.2** tab.
5. Select the **Supports Sync Level 2 Protocols** check box.

See Also
Concepts
Creating and Managing Remote Environments Using TI Manager
Getting Started with TI

Specifying SNA Attributes for Remote Environments

To determine the attributes that are required by Transaction Integrator (TI) when you are creating a new remote environment, see your Host Integration Server configuration. To obtain the required attribute values, open Host Integration Server SNA Manager or contact your Host Integration Server system administrator.

Note

To use two-phase commit, each local and remote logical unit (LU) must have SyncPoint support enabled in the SNA server n ode in which it is defined and should point to the computer that is running Resync services. For more information, see Providing a Fail-Safe Environment for ACID Transactions.

Local LU Alias

Enter the LU alias for the local APPC LU defined for the TI remote environment. From Host Integration Server SNA Manager, select the Host Integration Server computer that provides the required host connectivity. Then open the Local APPC LUs folder to obtain a list of configured local APPC LU aliases. Use a name from the list to specify this attribute.

Remote LU Alias

Enter the LU alias for the remote APPC LU defined for the TI remote environment. From Host Integration Server SNA Manager, select the computer running Host Integration Server that provides the required host connectivity. Then open the Remote APPC LUs folder to obtain a list of configured remote APPC LU aliases. Use a name from the list to specify this attribute.

Mode Name

Enter the APPC mode used for the connection. From Host Integration Server SNA Manager, open the APPC Modes folder to obtain a list of configured modes. Use a name from the list to specify this attribute.

To use two-phase commit, the APPC mode must be Sync Level 2 capable.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager

How to Assign a TI Component to a Remote Environment

Each TI component is associated with a remote environment (RE) type. As you add multiple REs of the same type, verify that each component is assigned to an RE of the type for which it is configured.

To assign a component to a specific RE of the appropriate type, you can use one of the following methods:

- Set a default RE for the Transaction Integrator (TI) components you create. When you deploy the TI component in a COM+ application , the component is automatically associated with the default RE of the appropriate type. Unless you specifically set the default RE, the default RE is the first RE of that type (or class) that you defined.

- Manually move components from one specific instance of an RE type to another specific instance of the same type. This action applies to TI components that you have already deployed in a COM+ application. You can use TI Manager to move components from one RE to another.

- Deploy the TI component by using HIS Designer. In HIS Designer, you are asked to associate the TI component with a specific RE. For more information about this option, see How to Deploy a TI Component.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

Working with Unassigned Components

Before a Transaction Integrator (TI) component appears in the console tree of TI Manager, it must be deployed in a COM+ application. Before you can successfully run an application that calls the component, you must associate the component with a remote environment (RE) that describes the region on the mainframe where the host transaction program resides. Any TI component that is not associated with an RE appears in the **Unassigned Components** folder of TI Manager. Such components are valid COM+ components. However, they cannot be used by your application until they are assigned to the proper RE.

A TI component appears in the **Unassigned Components** folder if you do not set the default registration RE. This can happen for either of the following reasons:

- You created the TI component and added it to a COM+ application, but you have not yet created an instance of an RE of the RE type required by the TI component.

- The default registration RE is set to <none> on the **Registrar** tab of the Transaction Integrator properties. This setting is useful, for example, if you do not want to assign components to an RE until the mainframe is fully configured, or until you are ready to test or deploy your TI application.

See Also
Concepts
Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Set a Default Remote Environment

Each Transaction Integrator (TI) component must be associated with a remote environment (RE) type. When you register a TI component by adding it to a COM+ application , the component is associated with the first instance of the same type that you defined using TI Manager. For example, if you create an instance of a CICS using LU 6.2 RE with the default name of CICS1, any component created for use with a CICS using LU 6.2 RE will be associated with CICS1. If you define more REs of the same type (for example, CICS2 and CICS3), you can specify the default RE for associated components that you later deploy.

If you want to associate a component with an RE at a later time, you can specify a default RE of none. Registered components with no RE specified are shown in the **Unassigned Components** folder in TI Manager.

To set a default RE

1. Start TI Manager.
2. Right-click **Transaction Integrator** in the console tree, and then click Properties.
3. Click the **Registrar** tab.
4. Click an RE type, and then click **Change**.
5. In the list, click the name of the RE that you want to be the default RE instance for that RE type.
6. Click **OK**.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Move a TI Component to Another Remote Environment

Each Transaction Integrator (TI) component is associated with a specific remote environment (RE), such as CICS Using LU 6.2, when you create the component in HIS Designer. Windows automatically registers a TI component with its appropriate RE when you add the TI component to a COM+ application to create a TI Automation server.

You can move a component to another RE when, for example, you want to move the component to a different region of a mainframe as your application goes through the cycle of development, testing, and deployment. TI components are designed for a certain type of RE, so you can only move a TI component to another RE of that identical type. However, you can move any component into the Unassigned Components folder to save it. Later, you can associate a component in the Unassigned Components folder with an appropriate RE. Use TI Manager to move TI components from one RE to another or to the Unassigned Components folder.

Note
You cannot move components across computer boundaries. For example, if you have two TI Manager consoles open on your computer, one for your computer and the other for a remote computer where TI is also installed, you cannot move a component from a RE on one console into a RE on the other console.

To move a TI component in TI Manager

1. Start TI Manager.
2. Under **Transaction Integrator** in the console tree, find the component that you want to move in either its current RE folder or in the Unassigned Components folder.
3. Click the component name, and on the **Action** menu, click **Move**.
4. In the **Move Component** dialog box, click a new RE instance name in the list. (The list only displays those currently defined REs that the selected component can be moved into.)
5. Click **OK**.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Locate a TI Component

The more remote environments (RE) that you have configured, the more difficult it can be to locate a specific Transaction Integrator (TI) component in TI Manager's console tree. You can, however, easily locate a specific component without expanding all the REs in the console tree.

To locate a specific TI component

1. Start TI Manager, right-click **Transaction Integrator** in the console tree, and click **Find Component**.
2. In the list, click the name of the component that you want to locate, and then click **OK**.

   The console tree shows the remote environment with which the component is associated.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Activate or Deactivate a Remote Environment

When a region on a mainframe fails or is taken offline for administrative maintenance, any method invocation for components associated with the remote environment (RE) that describes that region will fail. Therefore, when a mainframe region is unavailable, you should deactivate the RE that is supported by that region. You can then temporarily move the affected Transaction Integrator (TI) components to another RE (for example, a back-up region for the offline one) so that your TI applications can continue to run. When the mainframe region is restored, you can once again activate the RE for that region. Deactivating an RE for an offline region on a mainframe also reduces the number of error messages that are sent to the Windows Event Log.

You can check the status of REs by clicking the Remote Environments folder in the TI Manager console tree. The details pane shows icons for all the remote environments contained in the folder. These icons indicate whether a remote environment is active or inactive.

Note
If a transaction is in progress at the time that the region on the mainframe is taken offline, the transaction finishes. However, subsequent method invocations will fail.

To activate or deactivate a Remote Environment

Use one of the following methods to activate or deactivate an RE:

- Start TI Manager, right-click the remote environment (RE) for the mainframe region that you want to activate or deactivate, and click **Activate** or **Deactivate**.

  -or-

- Start TI Manager, click the remote environment (RE) for the mainframe region that you want to activate or deactivate, and then click the green arrow on the tool bar to activate an inactive RE, or click the red arrow to deactivate an active RE.

If you click the Remote Environments folder in the console tree, the details pane lists all of the REs and indicates the current state of each RE.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Delete a TI Component from a Remote Environment

Deleting a Transaction Integrator (TI) component from a remote environment (RE) deletes that component from the COM+ application. You can delete only one component at a time.

To delete a TI component from an RE

1. Start TI Manager.
2. In the console tree, double-click **Transaction Integrator** for the appropriate computer, double-click **Remote Environments**, and then double-click the RE that contains the component to be deleted.
3. Right-click the component to be deleted, and click **Delete**.
4. Click **Yes** to delete the selected component.

   To reinstate the component, add it back into the COM+ application.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Delete a Remote Environment

You can delete a remote environment (RE) when, for example, you need to move a mainframe transaction program (TP) from one region of the mainframe to another. When you delete an RE, all Transaction Integrator (TI) components assigned to that RE are placed in the Unassigned Components folder in TI Manager.

To delete a remote environment

1. Start TI Manager, and right-click the RE that you want to delete.
2. Click **Delete**.
3. Click **Yes** to confirm the deletion.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

How to Set or View Remote Environment Properties

Basic information about a remote environment (RE) is displayed on an item's property sheet. Property sheets are available for the following items:

- The Transaction Integrator (TI) folder
- An RE of any type
- Any component associated with an RE
- Any component present in the Unassigned Components folder

In addition, Context-sensitive Help is available for all properties.

Note
By default, no time-out value is specified when you create REs that use LU 6.2 or TCP/IP protocols. The TI run-time environment waits indefinitely for the mainframe transaction program to return output parameters. While waiting, the TI run-time environment also blocks the calling client application until a response is received. This behavior is typical for APPC applications. To avoid indefinite blocking, you can set a time-out value (in seconds) for REs using LU 6.2 or TCP/IP protocols. You set the value on the LU 6.2 or TCP/IP tab of the RE's properties page.

To view or set an item's properties

1. In TI Manager, right-click the item whose properties you want to view or set.
2. Click **Properties**.
3. If the property sheet has more than one tab, click the tab that contains the properties you want to set or view.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

Supporting Two-Phase Commit in a Remote Environment

Only the SNA protocols support two-phase commit (2PC). The TCP/IP protocol does not support 2PC. For an SNA remote environment (RE) to support 2PC, it must be set to support ACID (atomic, consistent, isolated, durable) transactions. To accomplish this, you must set the RE to support the Sync Level 2 protocol.

Transaction Integrator (TI) supports Sync Level 2 for CICS Using LU 6.2, for CICS LINK Using LU 6.2, and for IMS Using LU 6.2 REs. Sync level 2 is the default for all CICS REs but not for IMS REs where IMS version 6.0 and IBM's Resource Recovery Services (RRS) are required for Sync Level 2 support.

To activate Sync Level 2 support, follow the procedure in
How to Define a Transactional SNA CICS or SNA IMS Remote Environment.

Note
The Host Integration Server LU 6.2 Resync TP Service only compares logs with CICS and IMS regions when the RE has been s et to support Sync Level 2.

See Also

Concepts

Creating and Managing Remote Environments Using TI Manager
Creating and Managing TI Components
Getting Started with TI

Managing Transaction Integrator with TI Manager

Use TI (Transaction Integrator) Manager to create and manage remote environments (RE) for each mainframe region.

In This Section

How to Add or Remove TI Manager for a Remote Computer

You can use TI (Transaction Integrator) Manager to administer remote environments (RE) on your local client computer if you installed Host Integration Server Server on that computer. However, TI cannot reside on a computer that has Host Integration Server Administrator Client or Host Integration Server End-User Client installed instead of Host Integration Server Server.

Once you install Host Integration Server Server on the computer, you can use TI Manager to deploy TI components in COM+ applications on the local computer.

You can display a TI Manager on your local computer for your local computer and for up to nine remote computers that are currently running TI. This means that, from a single computer, you can administer the remote environment configurations on a maximum of 10 computers.

To add TI Manager consoles for remote computers to your local computer

1. Open Microsoft Management Console (MMC) by doing one of the following:
    a. Click **Start**, click **Run**, type **mmc**, and then click **OK**.
    b. Type **mmc** at a command prompt, and then press ENTER.
2. On the **Console** menu, click **Add/Remove Snap-in**.
3. In the **Add/Remove Snap-in** dialog box, click **Add**.
4. In the **Add Standalone Snap-in** dialog box, double-click **TI Manager**.
5. In the **Specify computer to administer** dialog box, click **Another Computer** and type the name of that computer (for example, mycomputer2).
6. If needed, select the **Allow the selected computer** check box so that in the future, you can specify a different computer when you start TI Manager from the command line. To learn how to do this, see How to Start TI Manager.
7. Click **Finish**.
8. Repeat steps 4–7 for each remote computer that you want to administer from your local computer. You can add a maximum of nine.
9. In the **Add Standalone Snap-in** dialog box, click **Close**.
10. In the **Add/Remove Snap-in** dialog box, click **OK**.

To remove a TI Manager console

1. On the TI Manager master menu, click **Console**.
2. Click **Add/Remove Snap-in**.
3. Click the TI console that you want to remove, click **Remove**, and then click **OK**.

See Also

Concepts

Managing Transaction Integrator with TI Manager
Getting Started with TI

How to Create Multiple Views of a Single TI Manager Console

Each instance of a TI (Transaction Integrator) Manager console shows the remote environments configured for a single computer. You cannot load more than one TI Manager console for a single computer. However, you can create multiple views of a single TI Manager console, and then display each view in a separate window. Typically, you can use this feature to display the different details for different selected items in the console tree. To save space on the screen, you can hide the console tree on each window.

When you add a new window, that window is superimposed on the original window. To make them all visible, cascade or tile the windows. You can then resize or reposition the windows.

To create multiple views of a TI console

1. Start TI Manager for the computer.
2. On the **Action** menu, click **New Window from Here**.
3. Repeat step 2 for the number of windows that you want to add.
4. On the **Window** menu, click either **Cascade** or **Tile Horizontally**, depending on how you want to configure the windows.
5. For each window, select the item in the console tree that you want to display.

To hide the console tree for a window

- On the window's **View** menu, click **Customize**.
- Clear the **Console tree** check box, and select the **Description bar** check box, and then click **OK**.

  The description bar describes the content of the remaining pane, so you no longer need the console tree.

See Also
Concepts
Managing Transaction Integrator with TI Manager
Getting Started with TI

Refreshing the TI Manager Display

The contents of TI (Transaction Integrator) Manager are automatically refreshed as new TI-related entries are written to the Windows Registry. For example, TI Manager refreshes if you add a TI component to, or delete one from, a COM+ application. An automatic refresh, however, does not sort the updates that it makes to the display. For example, if you create a new remote environment (RE), the new remote environment is added to the bottom of the RE tree, rather than inserted alphabetically within the tree. You can manually refresh the display to resort its contents. Manually refreshing the display is also the best way to ensure that TI Manager shows current data from the Windows Registry.

You can refresh the TI Manager display in either of two ways:

- On the **Action** menu, click **Refresh**.

-or-

- On the MDI toolbar, click **Refresh**.

See Also
Concepts
Managing Transaction Integrator with TI Manager

Creating and Managing TI Components

The topics in this section give you the information you need for creating and managing Transaction Integrator (TI) components.

In This Section

Reserved Words

When Host Integration Server Designer (HIS Designer) asks you to supply a name (for example, a method or parameter name), do not use any of the following reserved words. Words on this list are not case sensitive; for example, Byte, byte, and BYTE are all reserved words.

- Boolean
- BSTR
- Byte
- Currency
- CurrentRE
- Date
- Decimal
- Double
- enum
- Float
- IDispatch
- Int
- Integer
- Interface
- LastSM
- Long
- LPSTR
- LPWSTR
- module
- NewRecordset
- Password
- RECollection
- REStatus
- SelectionHint
- ServerPID

- Short
- Single
- String
- Struct
- typedef
- UserID
- VARIANT
- Void

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Create a New TI Component

Use Host Integration Server Designer (HIS Designer) to create a new Transaction Integrator (TI) component and populate it with methods.

To create a new TI component

1. Start HIS Designer.
2. On the **File** menu, click **New**.
3. Type a library name for the new TI component library.
4. Type a name for the TI component library's interface.
5. Type a version number for the component library.

   The number to the left of the decimal point is the major version number. The number to the right of the decimal point is the minor version number.
6. Click the type of remote environment (RE) that your component will use.
7. Click the component library's Transaction support property.

   In this case, you are deciding whether the component will support ACID (atomic, consistent, isolated, durable) transactions that each support two-phase commit (2PC). If you click **Does not support transactions**, your TI component will still support mainframe transactions in mainframe transaction programs (TP).
8. If you are providing Help for your application, type the context ID for the first topic in the **Starting Help Context ID** box.
9. Click **OK**.

   You will now see folders and icons for **Interface**, **Methods**, **Recordsets**, and **User-Defined Types**.

   To add a method to the interface

   - In the console tree of HIS Designer, right-click **Methods**, point to **Insert Method**, and then click a data type that represents the return value.

   To add a parameter to a method

   - In the console tree of HIS Designer, right-click the name of the method, point to **Insert Parameter**, and then click a data type.

   To add a recordset to the interface

   - In the console tree of HIS Designer, right-click **Recordsets**, and then click **Insert Recordset**.

   To add a member to the recordset

   - In the console tree of HIS Designer, right-click the name of the recordset, point to **Insert Recordset Member**, and then click a data type.

   To add a user-defined type to the interface

   - In the console tree of HIS Designer, right-click **User-Defined Types**, and then click **Insert User-Defined Type**.

   To add a member to the user-defined type

   - In the console tree of HIS Designer, right-click the name of the User-Defined Type, point to **Insert User-Defined Type Member**, and then click a data type.

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Import a TI Component

You can import the methods, recordsets, and user-defined types from an existing component library to another component library. This is useful, for example, if you want to build a new component library based on an existing one.

Imported methods, recordsets and user-defined types are added to those of the library into which they are imported. If duplicate method names exist between the two libraries, you are asked to supply a new name for a method before it is imported. Duplicate recordset names are allowed unless columns within the recordsets do not match, or Automation data types and associated COBOL data types within a column do not match. In this case, you are asked to supply a new name for a recordset before it is imported.

To Import a TI Component

1. In the console tree of HIS Designer, right-click the icon for your component library's interface.
2. Point to **Import**, and then click **Component Library**.
3. In the **Insert Component Library** dialog box, locate and click the TI component library (the .tlb file) that you want to import.
4. Click **Open**.

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Import COBOL into a TI Component

You can import COBOL source code to define the Automation interface of a new Transaction Integrator (TI) component library. To do so, use the TI COBOL wizard to create one method at a time. The COBOL wizard initially imports an entire source file for a mainframe transaction program (TP). As you step through the wizard, you extract the data declarations that describe input sent to, and output received from, the mainframe TP. These data declarations are used to define your TI component library. All other content in the source file is ignored.

As you develop your TI application, you can continue to use the COBOL wizard to make adjustments to a component definition. For example, you can add a parameter to a method to incorporate updates that have been made to the COBOL source code on the mainframe. To adjust a component definition, you can re-run the COBOL wizard to replace a method or a recordset in your component definition. Before replacing an existing method or recordset, you must unlock the method or recordset. Component definitions are locked by default to protect against unintended changes.

To import COBOL into a TI component

1. In the console tree of Host Integration Server Designer (HIS Designer), right-click the icon for your component library's interface.
2. Point to **Import**, and click **COBOL Wizard**.
3. Follow the instructions on the screen.

   The COBOL wizard steps you through the process of importing the COBOL source code that you want.

To unlock a method or record set

1. Start HIS Designer, and right-click the method or recordset.
2. Click **Unlock**.

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Export COBOL from a TI Component

You can use a component definition to generate COBOL syntax for the data declarations that describe input sent to, and output received from, the mainframe transaction program (TP). The generated COBOL syntax is saved in a text file. The file's contents are not an actual program, but data declarations. The file contains data only, not logic, and is intended to serve as a guideline, for example, for code that can be incorporated into a mainframe TP.

To export COBOL from a TI Component

1. In the console tree of HIS Designer, right-click the icon for your component library's interface.
2. Point to **Export**, and then click **Generate COBOL Declarations**.

   If you are exporting from a new (unsaved) component library, you are prompted to save the library.
3. Click **OK**.

   The **Save File As** dialog box appears.
4. Fill in the dialog box and then click **Save**.

   Transaction Integrator (TI) displays the text file in Notepad for you to view.

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

Adding TI Components to COM+ Applications

To deploy a Transaction Integrator (TI) component, add it to a COM+ application.

A TI component consists of a type library (.tlb file) that is created by using Host Integration Server Designer (HIS Designer). Typically, you deploy all components that are required for your Automation server application in a single COM+ application. There are no other files (such as associated DLLs) that you must have to deploy together with a TI component library. When you add the TI component library in a COM+ application , that COM+ application becomes an Automation server that is automatically associated with the generic TI run-time environment code (tagen.dll).

You can deploy a TI component using one of these methods:

- Using HIS Designer.
- Using TI Manager.
- Just dragging and dropping the component.

To use any of these methods, you must have administrative privileges.

After you deploy a TI component, you can view the component's interfaces and methods. Check the deployed component's property settings to verify that it is associated with the correct remote environment. To do this, right-click the component's name in the console tree, and then click **Properties**.

See Also
Concepts
Creating and Managing TI Components

How to Remove a TI Component from a COM+ Application

To remove a TI component from a COM+ application

1. Start TI Manager.
2. In the console tree, double-click **Component Services** (or **Microsoft Transaction Server**).
3. Double-click **Computers**, and then double-click the computer where the component you want to remove is located.
4. Double-click **COM+ applications**, double-click the COM+ application that contains the component, and then double-click its **Components** folder.
5. Right-click the component that you want to remove, and then click **Delete**.
6. Click **Yes** to confirm that you want to delete the selected component.

   When you remove a TI component from a COM+ application , you also remove the entry for the component from the Windows Registry. In addition, the representation of the component is removed from TI Manager. You can no longer use that component.

See Also
Concepts
Creating and Managing TI Components
Transaction Integrator User's Guide

How to Set or View Component Properties

A property sheet displays basic information about a Transaction Integrator (TI) component. Property sheets are available in Host Integration Server Designer (HIS Designer) for the following:

- Interface.
- Each method in an interface.
- Each parameter of a method.
- Each recordset in an interface.
- Each column of a recordset.
- Each user-defined type in an interface.
- Each member of a user-defined type.

To view or set an item's properties

1. In HIS Designer, right-click the item, and then click **Properties**.

Note
After you deploy a TI-created component in a COM+ application, that component acquires additional properties. For a complete view of a component's properties, use TI Manager to view the properties for the component and the properties for the COM+ application where the component is located.

Some properties are informational only, so you cannot change them. However, there are many properties that you can change. When you are viewing properties in HIS Designer, you can protect against inadvertently changing a component definition by locking the definition. Doing so makes property settings read-only until you unlock them.

To lock or unlock a component definition in HIS Designer

1. Start HIS Designer, and open a TI component.
2. In the console tree, right-click the component's interface, and then click **Lock** or **Unlock**.

   Context-sensitive Help is available for all properties.

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Set a TI Component's Transaction Property

Set a Transaction Integrator (TI) component's Transaction support property to tell COM+ whether that TI component supports COM-based ACID (atomic, consistent, isolated, durable) transaction processing. When an instance of a TI component is created, COM+ always checks the component's transaction property to determine whether the instance should run in a COM-based ACID transaction.

Note
The term *transaction* can be confusing because it can mean different things depending on the context. A mainframe transacti on program can contain one or more mainframe transactions. However, a mainframe transaction is not necessarily an ACID t ransaction.

Not all components are designed to support COM-based ACID transaction processing. If your component is not designed to participate in an existing COM+ transaction or start a new COM+ transaction, be sure to set the component's Transaction support property to **Does not support transactions** in Host Integration Server Designer (HIS Designer) or to **Not supported** in TI Manager. This does not mean that the TI component does not support mainframe transactions; it means that the component does not support COM-based ACID transaction processing.

The application developer initially sets a component's transaction property in HIS Designer. After a TI component is deployed in a COM+ application , an administrator can change a TI component's transaction property to accommodate changing needs by using TI Manager or HIS Designer.

In HIS Designer, you can set the Transaction support property to one of the following values:

- **Requires a transaction** This TI component is to be used in an application that executes within a COM+ ACID transaction. This TI component's methods are called by client applications that must automate mainframe transaction programs (TP) that support sync level 2 requests (also known as two-phase commit). If a transaction is in progress, the application is enlisted in the transaction. Otherwise, COM+ starts a new transaction. None of the TCP/IP-based remote environments (RE) support this property because TCP/IP does not support two-phase commit. For an IMS Using LU 6.2 RE to support this property, you must be using IMS version 6.0 or later. The CICS LINK Using LU 6.2 RE type supports this property. The CICS Using LU 6.2 RE type requires mainframe ACID transaction processing support.

- **Requires a new transaction** This TI component is to be used in an application that executes within a COM+ ACID transaction. This TI component's methods are called by client applications that must automate mainframe transaction programs (TPs) that support sync level 2 requests (also known as two-phase commit). By setting this property, you are telling COM+ to always start a new ACID transaction, regardless of whether or not an existing ACID transaction is already in progress. None of the TCP/IP-based remote environments (RE) support this property because TCP/IP does not support two-phase commit. For an IMS Using LU 6.2 RE type to support this property, you must be using IMS version 6.0 or later. The CICS LINK Using LU 6.2 RE type supports this property. The CICS Using LU 6.2 RE type requires mainframe ACID transaction processing support.

- **Supports transactions** This TI component is to be used in an application that may or may not execute within a COM+ ACID transaction. This TI component's methods are called by client applications that must automate mainframe transaction programs (TPs) that support both sync level 0 (nontransactional) mainframe transactions and sync level 2 (transactional two-phase commit) mainframe transactions. For a sync level 2 transaction, if a transaction is in progress, the client application is enlisted in the transaction. Otherwise, COM+ starts a new transaction. None of the TCP/IP-based remote environments (REs) support sync level 2 because TCP/IP does not support two-phase commit. For an IMS Using LU 6.2 RE to support sync level 2, you must be using IMS version 6.0 or later. The CICS LINK Using LU 6.2 RE type supports this property. For the CICS Using LU 6.2 RE type to support sync level 2, the mainframe must support ACID transaction processing.

- **Does not support transactions** This TI component is to be used in an application that does not execute within a COM+ transaction. This component's methods are called by applications used with mainframe TPs that support sync level 0 mainframe transactions. All of the TCP/IP and SNA remote environment (RE) types support this property.

In TI Manager, in addition to these four possible transaction properties, you can disable transaction support for a TI component. The five values for the Transaction support property in TI Manager are as follows:

- **Disabled** Click this to eliminate ACID transaction-related overhead for objects that will never need access to a resource manager. This attribute simulates the transactional behavior of an undeployed component (a COM component that has not been installed in a COM+ application).

- **Not Supported** Click this to prevent an object from participating in an ACID transaction, regardless of the transactional status of its caller. Declaring this value guarantees that an object will not vote in its caller's ACID transaction or begin an ACID transaction of its own. This is the default value for all components. This is equivalent to the **Does not support transactions** setting in HIS Designer.

- **Supported** Click this to allow an object to participate in an ACID transaction if one exists. Declaring this value causes an object to share in its caller's ACID transaction but prevents it from initiating an ACID transaction of its own. This is equivalent to the **Supports transactions** setting in HIS Designer.

- **Required** Click this to specify that all objects created from the component will be transactional (that is, all must meet the ACID test). This is the preferred setting for a object that performs resource activities because it guarantees transaction protection for those activities. This is equivalent to the **Requires a transaction** setting in HIS Designer.

- **Requires New** Click this to require that an object be the root of a new transaction, regardless of the transactional status of the caller. COM+ automatically initiates a new transaction, which is distinct from the caller's transaction. This is equivalent to the **Requires a new transaction** setting in HIS Designer.

Use the following procedures to set or change the transaction property for a TI component.

- To set the transaction property for a new TI component

- To change an undeployed TI component's transaction property in HIS Designer

- To change a deployed TI component's transaction property in TI Manager

To set the transaction property for a new TI component

1. Start HIS Designer.
2. On the **File** menu, click **New**.
3. Click the transaction property value that you want under **Transaction Support in the New Component Library** dialog box.

To change an undeployed TI component's transaction property in HIS Designer

1. Open the TI component in HIS Designer.
2. In the console tree, right-click the TI component's interface, and then click **Properties**.
3. Under **Transaction support**, click the value you want to assign to the component, and then click **OK**.

To change a deployed TI component's transaction property in TI Manager

1. Start TI Manager.
2. In the console tree, under the **Component Services** (or **Microsoft Transaction Server**) folder, browse to the COM+ application where the TI component is located.
3. Right-click the name of the TI component, and the click **Properties**.
4. Click the **Transactions** tab, click one of the five options under **Transaction support**, and then click **OK**.

See Also
Concepts
Creating and Managing TI Components
Transaction Integrator User's Guide

Managing TI Calls Using Status and Timeout Properties

A client application can manage its calls to a Transaction Integrator (TI) Automation server by checking the TI component's remote environment (RE) **Status** property and the **Timeout** property.

Status Property

TI provides a read-only RE **Status** property in each component library created by Host Integration Server Designer (HIS Designer). A client application can use this property to inquire about the current state of the RE with which a TI component is associated. It returns whether the RE is enabled, disabled, or blocked by a communications difficulty.

Timeout Property

All RE types supported by TI include a **Timeout** property. Set the **Timeout** property value on the **LU 6.2** or **TCP/IP** tab of the remote environment's properties page in TI Manager.

By default, an RE has no initial **Timeout** property value. Therefore, unless you use TI Manager to set a **Timeout** value, the TI run-time environment waits indefinitely for the mainframe transaction program (TP) to return output parameters. Meanwhile, the TI run-time environment blocks the calling client application until this response is received. This blocking behavior is typical for APPC applications.

For example, with LU 6.2, if an IMS program is disabled, request messages continue to be placed successfully on the IMS message queue without network errors being reported. This occurs even when these messages are not being processed.

Set the **Timeout** value to free a blocked client application after the time-out interval expires. After the time-out period expires, the client application is notified that a time-out error occurred when attempting to execute the IMS program. However, because the requests are successfully stored in the IMS message queue, the requests can still be processed later if the IMS program is enabled without first emptying the IMS queue.

Use TI Manager to specify a **Timeout** value, in seconds, for a given remote environment. Right-click the RE, and then click **Properties**.

Handling Time-out Errors

When sending messages to the CICS or IMS region described by a specific RE, the TI run-time environment measures the amount of elapsed time that occurs from when a request is sent to when a response is received. If the time-out interval elapses before receiving a response, the TI Automation server object is terminated, and the associated COM+ transaction stops the transaction and reports the error to the client application. A message describing this error is also written to the Windows Event Log.

To handle a time-out error, the TI run-time environment unbinds the LU 6.2 session established with the CICS or IMS region. This means that the TI run-time environment must reestablish a new LU 6.2 session before another message can be sent to this region. If the time-out error occurs over a TCP/IP connection, TI shuts down the TCP/IP connection.

Time-out errors can adversely affect the performance of TI. Therefore, you should set time-out values high enough to signal a significant failure in the remote CICS or IMS region.

Note
For TCP/IP, the time-out value set on an RE's properties page is significant only to the sending and receiving of data. In contr ast, the time-out value for establishing the connection itself is defined by the implementation of the underlying TCP transport .

See Also

Concepts

Creating and Managing TI Components

Transaction Integrator User's Guide

How to Print a Component Description

Use the **Print** command in Host Integration Server Designer (HIS Designer) to print a description of the current component. This printed description includes the following:

- A description of the component's interface properties.
- A description of each method, its parameters, and its properties.
- A description of each recordset, its columns, and its properties.

To use the Print command

- In HIS Designer, click **Print** on the **File** menu.

See Also
Concepts
Creating and Managing TI Components
Transaction Integrator User's Guide

How to Run TI Over TCP/IP

You can install and run Transaction Integrator (TI) over TCP/IP without installing or using any of the SNA services of Host Integration Server.

Use the following procedure to run a TI application over TCP/IP.

To run a TI application over TCP/IP

1. Configure the mainframe (CICS or IMS) for TCP/IP, and establish a connection with your Windows-based Host Integration Server Server computer.

   For more information, see the following:

   - Configuring CICS for TCP/IP
   - Configuring IMS for TCP/IP.
   - Configure Host Environment and Programming Model Wizard Page in the New Remote Environment Wizard
   - Enhanced Listener CICS Administration

2. Install the COBOL programs within the CICS or IMS region that receives TI-initiated calls.
3. Define an appropriate TCP/IP remote environment for the CICS or IMS region that receives TI-initiated calls.
4. Build the TI component with a method for each transaction in the transaction program (TP).
5. Deploy the TI component in a COM+ application to create a TI Automation server.
6. Run the client application that calls the new TI Automation server to automate the TP.

See Also

Concepts

Configuring CICS for TCP/IP
Configuring IMS for TCP/IP
Defining a TCP/IP Remote Environment

Other Resources

Using Windows-Initiated Processing

Configuring CICS for TCP/IP

CICS TCP/IP Platform Requirements

TCP/IP version 3R2

CICS version 3.3 or later

Connections to CICS using TCP/IP

CICS uses the IBM-supplied Concurrent Listener (program EZACIC02, transaction ID CSKL) to establish an interaction with TCP/IP. The Listener is a transaction that automatically starts when CICS TCP/IP is started and enabled. When the Listener starts, it obtains a socket on which it can "listen" for connection requests from TCP/IP. The Listener binds the socket to a specified port, and then waits for a client request on that port. TCP/IP maintains a relationship of a port number to a CICS job. When a client makes a request on a port associated with CICS, TCP/IP forwards the connection request to the Listener in that CICS job.

For additional details about the CICS MS LINK communication model, see CICS MS LINK (TCP/IP).

TCP/IP-to-CICS Configuration

A TCP/IP port number is associated with a CICS region in the TCP/IP profile data set (hlq.PROFILE.TCPIP). The port statement is used to define this relationship. For example, the following is a port statement that associates port 3000 with CICS region CICSRG:

```
3000 TCP CICSRG
```

CICS-to-TCP/IP Configuration

The following sample host definition shows configuration parameters for CICS-to-TCP using the EZAC transaction:

```
EZAC,DEFINE
ENTER ONE OF THE FOLLOWING
CICS         ===> yes                 Enter Yes|No
LISTENER     ===>                     Enter Yes|No
EZAC,DEFINE,CICS
ENTER ALL FIELDS
APPLID       ===> CICSRG              APPLID of CICS System
EZAC,DEFINE,CICS
OVERTYPE TO ENTER
APPLID       ===> CICSRG              APPLID of CICS System
TCPADDR      ===> TCPIP               Name of TCP Address Space
NTASKS       ===> 020                 Number of Reusable Tasks
DPRTY        ===> 000                 DPRTY value for ATTACH
CACHMIN      ===> 015                 Minimum Refresh Time for Cache
CACHMAX      ===> 030                 Maximum Refresh Time for Cache
CACHRES      ===> 010                 Maximum number of Resolvers
ERRORTD      ===> CSMT                TD Queue for Error Messages
```

The following sample host definition shows configuration parameters for the CICS Concurrent Listener using the EZAC transaction:

```
EZAC,DEFINE
ENTER ONE OF THE FOLLOWING
CICS         ===>                     Enter Yes|No
LISTENER     ===> yes                 Enter Yes|No
EZAC,DEFINE,LISTENER
ENTER ALL FIELDS
APPLID       ===> CICSRG              APPLID of CICS System
NAME         ===> CSKL                TRANSACTION NAME OF LISTENER
EZAC,DEFINE,LISTENER
OVERTYPE TO ENTER
APPLID       ===> CICSRG              APPLID of CICS System
```

```
TRANID       ===> CSKL                Transaction Name of Listener
PORT         ===> 03000               Port Number of Listener
IMMEDIATE    ===> YES                 Immediate Startup   Yes|No
BACKLOG      ===> 010                 Backlog Value for Listener
NUMSOCK      ===> 050                 Number of Sockets in Listener
MINMSGL      ===> 004                 Minimum Message Length
ACCTIME      ===> 060                 Timeout Value for ACCEPT
GIVTIME      ===> 030                 Timeout Value for GIVESOCKET
REATIME      ===> 000                 Timeout Value for READ
FASTRD       ===> YES                 Read Immediately    Yes|No
TRANTRN      ===> YES                 Translate TRNID     Yes|No
TRANUSR      ===> YES                 Translate User Data Yes|No
SECEXIT      ===>                     Name of Security Exit
```

Before you attempt to use the TCP/IP connection, do the following:

- Verify that you have a TCP address space running on the host. (You should be able to PING the host at its IP address or DNS name.) Record the IP address; you will need to know it later when you use Transaction Integrator (TI) Manager to define a TCP/IP remote environment for the CICS region.

- Check that the CICS region supports TCP/IP, and that the IBM-supplied Listener (program EZACIC02, transaction ID CSKL) is defined. These procedures are described in chapter 5 of *TCP/IP V3R2 for MVS: CICS TCP/IP Socket Interface Guide* (IBM Document #SC31-7131). Note that this is a CICS TS version 1.2 document, but the configuration is also supported in CICS version 4.1.

- Determine the IP port number of the Listener (EZAC DISPLAY LISTENER); you will need to know it when you use TI Manager to define a TCP/IP remote environment for the CICS region.

- Start the IBM-supplied Listener (EZAO START) and check the CICS view of the Listener status (execute the CEMT INQUIRE TASK command, and verify that CSKL is running).

See Also

Tasks

How to Run TI Over TCP/IP

Configuring IMS for TCP/IP

This section describes the necessary steps in configuring IMS for TCP/IP. It may also be necessary to set up and configure the Host Web Service. See your IBM documentation for information on this.

IMS TCP/IP Platform Requirements

- TCP/IP version 3R2
- IMS version 4 or later

Connections to IMS using TCP/IP

IMS uses a Listener (program EZAIMSLN) to establish an interaction with TCP/IP. The Listener in an IMS Batch Message Processing (BMP) helps facilitate the connection process. When the Listener starts, it obtains a socket on which it can "listen" for connection requests from TCP/IP. The Listener binds the socket to a specified port, and then waits for a client request on that port.

TCP/IP maintains a relationship of a port number to an IMS Listener BMP. When a client makes a request on a port associated with IMS, TCP/IP forwards the connection request to the Listener in that BMP.

Implicit mode

Implicit mode uses the IMS Assist Module to translate conventional IMS communication into corresponding socket calls. The Implicit mode is dependent on the IBM-supplied default Listener (EZAIMSLN) that runs in a BMP region.

The host server application model processes input data using the IMS message queue. The Listener puts the TRANID and the input data into the queue. The IMS control region schedules the transaction in a Message Processing Region. The Transaction Program reads the request from the queue using GU and GN commands. All response data is delivered to the client by way of the ISRT command. The IBM-supplied Assist Module delivers the data directly to the client through socket API calls.

The Assist Module uses the DBLADLI API for Implicit mode. Host applications are written using CBLADLI or CBLTDLI APIs. If you want existing IMS applications to use Implicit Mode TCP/IP, you must change to the CBLADLI API and recompile the program.

Explicit mode

The IMS Explicit (TCP/IP) model requires the installation, within IMS, of the IBM-supplied default Listener (EZAIMSLN) that runs in a BMP region. This host server application model processes data without using the IMS message queue. The Listener places only a single segment (the Transaction Initiation Message) into the message queue. The IMS control region schedules the execution of the transaction into a Message Processing Region. The transaction then communicates directly with the client through socket API calls.

All IMS host server programs must be administered by IMS as no-response transactions.

For additional details about the IMS communication models, see IMS Implicit and IMS Explicit.

TCP/IP-to-IMS Configuration

A TCP/IP port number is associated with an IMS Batch Processing Region (BPR) in the TCP/IP profile data set (hlq.PROFILE.TCPIP). The port statement is used to define this relationship. For example, the following is a port statement associating port 3000 with an IMS batch region with a job name of WNWIBR1:

```
3000 TCP WNWIBPR1
```

IMS-to-TCP/IP Configuration

You can start an IMS Message Processing Program by specifying the program name of the IBM-supplied Listener program (EZAIMSLN). The Listener reads a configuration file identified by the DD statement **LSTNCFG**. This configuration data set contains one or more of the following startup parameter statements (one set for each transaction defined for at least one Command Region):

- **TCPIP** statement. Identifies the job name for the TCP/IP address space that manages the connection for the Listener
- **LISTENER** statement. Specifies the port number that the Listener will use. This statement also specifies other port-related

parameters, such as backlog, time-out values, and so on.

- **TRANSACTION** statement. Defines a list of transactions that the Listener can start. It also defines whether the Implicit or Explicit connection mode is used.

The Listener uses the three previously listed parameter statements to inform TCP/IP which port to use and which transactions can be accessed through TCP/IP.

The following is a sample of an IMS-to-TCP/IP host definition:

```
TCPIP ADDRSPC=WNWTCP31
LISTENER PORT=4000 BACKLOG=50
TRANSACTION NAME=TRANIMPL TYPE=IMPLICIT
TRANSACTION NAME=TRANEXPL TYPE=EXPLICIT
```

See Also

Tasks

How to Run TI Over TCP/IP

Defining a TCP/IP Remote Environment

The following table shows how you can define a TCP/IP remote environment (RE) to fit each of the five mainframe-based programming models even though there are only two Transaction Integrator (TI) RE types.

TI Remote Environment Type	Mainframe Programming Model
CICS and IMS using TCP/IP	CICS Concurrent Server (TCP/IP)
CICS and IMS using TCP/IP	CICS MS Link (TCP/IP)
CICS and IMS using TCP/IP	IMS Implicit (TCP/IP)
CICS and IMS using TCP/IP	IMS Explicit (TCP/IP)
IMS Connect or OTMA	IMS Connect or OTMA (TCP/IP)

Use TI Manager to define either TCP/IP RE.

See Also

Tasks

How to Run TI Over TCP/IP

Concepts

Transaction Integrator User's Guide

How to Run TI over SNA (APPC/LU 6.2)

If you have a distributed network, use Transaction Integrator (TI) over an SNA (APPC/LU 6.2) network connection to take advantage of its support for two-phase commit (2PC) in ACID (atomic, consistent, isolated, and durable) transaction processing. To support ACID transactions, your COBOL transaction program (TP) must support Sync Level 2. The TCP/IP protocol has no default support for 2PC, so TCP/IP is not appropriate in a distributed network.

To run a TI application over an SNA network

1. Configure the mainframe (CICS or IMS) for SNA, and establish a connection with your Windows-based Host Integration Server computer.
2. Install the COBOL transaction programs within the CICS or IMS region that receives TI-initiated calls.
3. Define an appropriate SNA remote environment for the CICS or IMS region that receives TI-initiated calls.
4. Build the TI component with a method for each transaction in the TP.
5. Deploy the TI component in a COM+ application to create a TI Automation server.

Run the client application that calls the new TI Automation server to automate the TP.

See Also

Tasks

How to Run TI Over TCP/IP

Other Resources

Using Windows-Initiated Processing

Defining an SNA Remote Environment

You can define an SNA (APPC/LU 6.2) remote environment (RE) instance to fit each of the following three mainframe-based programming models:

- CICS Link Using LU 6.2
- CICS Using LU 6.2
- IMS Using LU 6.2

Use TI Manager to define instances for any of these RE types.

See Also

Tasks

How to Run TI over SNA (APPC/LU 6.2)

Other Resources

Using Windows-Initiated Processing

Meeting Specific Real-World Needs

The topics in this section show you how to use Transaction Integrator (TI) together with COM+ to meet specific real-world application integration needs.

In This Section

Determining Who Initiated a Transaction

Providing a Fail-Safe Environment for ACID Transactions

Using TI in a Non-DPL Environment

See Also

Other Resources

Using Windows-Initiated Processing

Determining Who Initiated a Transaction

You can easily determine who initiated a specific transaction. This is helpful, for example, when you need to track down the history of a transaction failure. You can also use this technique to implement resource or transaction-level, per user, security.

When you select either user-level or package-level security on the **Security** tab of the Transaction Integrator (TI) Remote Environment (RE) properties page, TI sends security information in the session request to the host. You can deploy the Host Account Mapping database known as the Host Account Cache (HAC), and set up a mapping between each Windows user and the corresponding host user ID, and that is what TI will send. Or you can use the **Allow application to override security** option on the **Security** tab, and have the application return any host user ID (and password).

Whether the host will do anything with the different user IDs depends mostly on the ATTACHSEC setting for the CICS connection; this corresponds to the APPC LU that TI uses. The default ATTACHSEC setting is **local**, meaning that CICS does not validate the user ID in the session, and CICS runs the transaction in a default host credential. But if you set the ATTACHSEC setting, CICS uses Resource Access Control Facility (RACF) to validate the user ID in the session, and CICS then attaches that user ID to the trusted computing base (TCB) for the transaction as it runs through the Mirror transaction into the target mainframe transaction program.

See Also
Concepts
Meeting Specific Real-World Needs

Providing a Fail-Safe Environment for ACID Transactions

ACID (atomic, consistent, isolated, and durable) transaction processing using two-phase commit (2PC) generally requires a fail-safe environment. This is an environment that ensures continuation despite hardware failures. This is often called *2PC failover* or *hot backup*.

Host Integration Server includes enhancements to the SNA LU 6.2 Resync transaction program (TP) generally referred to as the *Resync service* together with enhancements to the configuration and APPC DLL to make 2PC failover work through two or more redundantly configured Host Integration Server SNA servers (computers). In the event of a failure of one of the servers (computers), a separate Host Integration Server computer running either Transaction Integrator (TI) or the DB2 Provider can continue to initiate transactions through an alternate server (computer).

Configuring 2PC Failover

To configure 2PC failover to work with Host Integration Server, complete the following tasks:

- Configure two Host Integration Server servers to support the same SyncPoint-enabled local APPC LU alias but with different LU names. Have these local APPC LUs point to the same computer name where Microsoft Distributed Transaction Coordinator (DTC) service and the Resync service are running (that is, a separate Host Integration Server computer that supports TI or an application that uses the DB2 Provider). Also, have both servers support the same remote APPC LU alias and name.

- In the applicable TI remote environment (RE), configure the local and remote LU aliases, and select transactional support. If the application is using the DB2 Provider, configure the Universal Data Link with the local and remote APPC LU aliases, and set the Units of Work property to **DUW**.

When the Resync service starts, it searches all SyncPoint-enabled local APPC LUs that specify the computer name where the Resync service is running. Resync then initiates an Exchange Log Names request over every found local APPC LU with all SyncPoint-enabled remote APPC LUs.

When a TI Automation server (application) or the DB2 Provider invokes a transaction program (TP) on the mainframe and initiates a conversation, the APPC DLL locates any available Host Integration Server server (computer) that supports the LU/LU pair. In this way, a TI Automation server (application) or the DB2 Provider gains fault tolerance by getting a conversation through any Host Integration Server server (computer) that supports the LU/LU pair. The Resync service then coordinates the DTC transaction log reconciliation when a Host Integration Server SNA server (computer) comes back online, if a server (computer) failure occurs during a transaction. Note that this configuration does not provide fault tolerance for the Host Integration Server server (computer) that is running only TI or the DB2 Provider, not the SNA service.

Note that clustering the servers (computers) that are running the SNA service is not recommended. Instead of using Windows Clustering, use the configuration recommendations described in this topic. In addition, 2PC works only when you are using the SNA (APPC/LU 6.2) protocol to communicate with the host system. Neither TI nor the DB2 Provider support 2PC over the TCP/IP transport, so there is no 2PC failover solution for TCP/IP-based systems.

See Also
Concepts
Meeting Specific Real-World Needs

Using TI in a Non-DPL Environment

A non-linked environment (that is, a non-DPL environment) is one that does not use IBM Distributed Program Link (DPL). You can use Transaction Integrator (TI) to invoke a mainframe transaction program (TP) that uses the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands. These two COBOL commands are useful when you want a CICS TP to take on SNA (APPC/LU 6.2) conversation responsibilities and therefore bypass the Mirror TP. In other words, the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands are most often used in a non-linked environment to transfer data to and from a logical unit (LU) of type 6.2 (APPC).

TI supports the LU 6.2 model for both linked and nonlinked environments. You can create the following remote environment (RE) types to support each model:

- **CICS Link using LU 6.2** Use this in an IBM DPL environment that uses the Mirror TP.
- **CICS using LU 6.2** Use this in a non-DPL environment that bypasses the Mirror TP.

Separating Business Logic from Presentation Logic

Many customers use TI in a non-DPL environment. The only concern is whether terminal logic is embedded with the business logic. When a COBOL TP supports IBM DPL, the presentation logic has already been separated from the business logic, so you probably will not need to modify the COBOL. However, if the TP was written to communicate with a terminal, it is likely that you will need to modify the COBOL code to separate the business logic from the presentation logic.

For example, the following sample COBOL code shows how to handle unbound recordsets by using the **EXEC CICS RECEIVE INTO** and **EXEC CICS SEND FROM** COBOL commands:

```
*****************************************************
* Example showing how to send unbounded recordsets
* to a client application.
*****************************************************
 IDENTIFICATION DIVISION.

 ENVIRONMENT DIVISION.

 DATA DIVISION.

 WORKING-STORAGE SECTION.

* INPUT AREA
 01  CUSTOMER-INPUT-NUMBER               PIC 9(9).

* OUTPUT AREA
 01  CUSTOMER-DATA.
     05  LAST-NAME                       PIC X(20).
     05  FIRST-NAME                      PIC X(20).

* ONE ROW IN A DATABASE TABLE
 01  INVOICES.
     05  INVOICE-NUMBER                  PIC 9(10).
     05  INVOICE-DATE                    PIC 9(7) COMP-3.
     05  INVOICE-AMOUNT                  PIC S9(13)V9(2) COMP-3.
     05  INVOICE-DESCRIPTION             PIC X(40).

 LINKAGE SECTION.

 PROCEDURE DIVISION.
*
*   Get the input customer account number from the
*   client applicaton:
*
     MOVE LENGTH OF CUSTOMER-INPUT-NUMBER TO RECEIVE-LENGTH
     EXEC-CICS RECEIVE INTO(CUSTOMER-INPUT-NUMBER)
               LENGTH(RECEIVE-LENGTH)
               END-EXEC.
```

```
*
*   Do some work; then send the first and last name back:
*
     MOVE LENGTH OF CUSTOMER-DATA TO SEND-LENGTH
     EXEC-CICS SEND FROM(CUSTOMER-DATA)
               LENGTH(SEND-LENGTH)
               END-EXEC.
*
*   Follow regular data with unbounded table data that
*   the client application sees as a recordset:
*
     MOVE LENGTH OF INVOICES TO SEND-LENGTH
     PERFORM UNTIL        NO-MORE-INVOICES
*
*   Do some work to get the next row:
*
     MOVE DB-INVOICE-NUMBER TO INVOICE-NUMBER
     MOVE DB-INVOICE-DATE   TO INVOICE-DATE
     MOVE DB-INVOICE-AMOUNT TO INVOICE-AMOUNT
     MOVE DB-INVOICE-DESC   TO INVOICE-DESCRIPTION
*
*   Send the row:
*
     EXEC-CICS SEND FROM(INVOICES)
               LENGTH(SEND-LENGTH)
               END-EXEC.
     END-PERFORM.
```

See Also

Concepts

Meeting Specific Real-World Needs

Using Host-Initiated Processing

The topics in this section explain how to use host-initiated processing.

In This Section

Creating a Local Environment

A local environment (LE) is a set of end points that the host-initiated processing (HIP) runtime uses to listen for incoming requests from the host. If communication with the host uses the TCP/IP network protocol, the endpoints are either port numbers or service names. If communication with the host uses the SNA network protocol, the endpoint is the local logical unit (LU) alias.

Use the **New Local Environment Wizard** to create a new local environment.

See Also

Other Resources

Using Host-Initiated Processing

Creating a Host Environment

A host environment (HE) defines the network and hardware characteristics of the non-Windows host that will initiate requests to the Windows operating system. For example, a host that uses the TCP/IP protocol to initiate requests to Windows is identified to host-initiated processing (HIP) through an HE that contains the IP address of the host and the code page that the host uses to represent its data.

You can use the **New Host Environment Wizard** to create a new host environment.

See Also

Other Resources

Using Host-Initiated Processing

Creating a Security Policy

You can create and manage security policy definitions in the **Security Policies** node in the HIP (host-initiated processing) Console.

When the HIP Console is first started, the **Security Policies** node is empty. The **Security Policies** node contains definitions for how Windows security credentials are established before the execution of the server object. The source of the security credentials can be the following:

- Based on User IDs and Passwords delivered to HIP by the client application program.
- Based on User IDs and Passwords delivered to HIP by well-defined host protocol standards (SNA attach header: FMH5).
- Default host-based User ID and Password.
- Windows credentials that the HIP application runs under

When host-based credentials are used, the Windows credentials are obtained by using the Single Sign-On (SSO) feature. This feature translates host-based User ID, Password and SSO Affiliated Application ID to a security identification number (SID) that is representative of the Window credentials. The server object is then executed with the translated security credentials.

You can create a new security policy by using the **Security Policy Wizard**.

See Also

Other Resources

Using Host-Initiated Processing

Defining New Objects

Objects in the host-initiated processing (HIP) environment represent the Windows Server that will be started after a request is received from a host. The object is defined by a Transaction Integrator Metadata (TIM) file that is generated by the development-time tool in Microsoft Visual Studio.

See Also

Other Resources

Using Host-Initiated Processing

Creating an Object View

Object views in host-initiated processing (HIP) provide different ways to view and manage Windows Server objects in the HIP environment. During the definition of an object view, all methods or a subset of the Objects methods can be defined in the view. This helps provide a level of security by limiting the number of methods available for execution.

When a host environment (HE) is associated with an object view, the level of security is elevated. With the HE association, only certain hosts can execute specific methods of an Object.

When the object view is finally associated with a local environment, the security is enhanced further by restricting the execution of a method on an object to a request from a specific host that made a request to a specific transport endpoint on the Windows operating system.

You can create new object views by using the **New Object View Wizard**. This wizard helps you construct the following administrative entities:

- A view of an object.
- One or more methods in the object to be defined on the view.
- A local environment association.
- Host environment association.
- Method resolution information.

See Also
Other Resources
Using Host-Initiated Processing

How to Modify Objects

After an object has been created and assigned to an application within the host-initiated processing environment, you must be careful when making changes to the object or its Transaction Integrator metadata (TIM) file.

To safely modify a TIM file for an existing object

1. In the TI Manager console tree, right click the object to be changed, and then click **Delete**.
2. Exit TI Manager.
3. Modify the TIM file and save it.
4. Start TI Manager.
5. In the TI Manager console tree, right-click **Objects**, point to **New**, and then click **Object**.
6. Follow the instructions in the **Object Wizard** to re-create the object.
7. In the TI Manager console tree, right-click the re-created object, point to **New**, and then click **View**.
8. Follow the instructions in the **New Object View Wizard** to re-create the object views.

Creating a New Application

Host-initiated processing (HIP) applications represent the execution environment for Windows Server objects that are initiated or driven from host requests. The HIP application execution environment is hosted in and synonymous with a Windows service.

A HIP application can host more than one server object. A HIP application can also have more than one listening endpoint associated with it.

Application definitions are created and managed in the **Computers** node on a specific computer. When the HIP Console is first started, the **Computers** node has only a single computer defined. That single computer is the computer where the HIP administrative console is running. When the Console is first started, no applications are defined on that computer.

You can add a new application to the Transaction Integrator (TI) Manager without specifying the local environment, host environment, objects, and object views used by the application.

See Also

Other Resources

Using Host-Initiated Processing

Transaction Integrator Performance Guide

To ensure excellent performance out of the box, Microsoft has run performance tests on Transaction Integrator (TI) to determine its performance and scalability in various deployment situations. This section discusses what was learned about TI.

Many factors can affect performance:

- Security.
- Whether you are using two-phase commit.
- Sharing the server with other Microsoft BackOffice applications.
- Sizing the server initially.
- Planning for future growth.

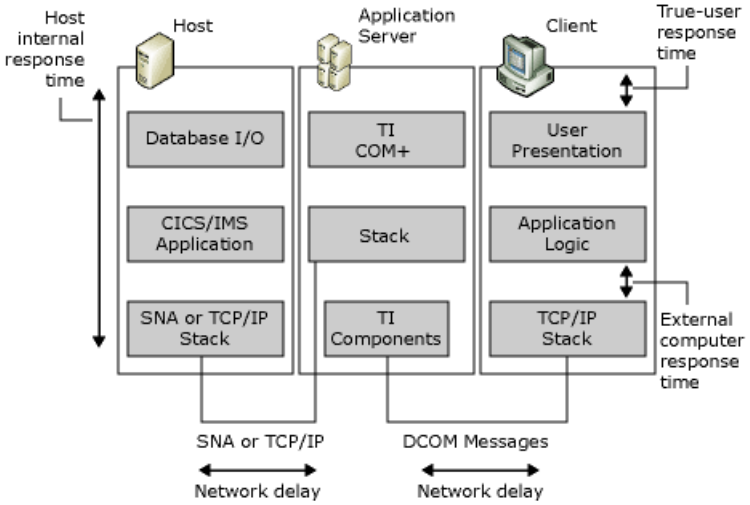
This section will help you handle various load conditions and help you tune your system to give the best possible TI performance while avoiding performance pitfalls, especially with transactions that run for a long time.

In This Section

Major Elements Affecting Overall Performance

Several factors contribute to the total user response time, often called the *response time budget*. To analyze the total system response time more easily, divide the response time into the parts of the system that play a major role in the total response time budget. The following figure shows the main divisions.

Three main divisions -- host, application server, and client --showing how response time is affected by at least three levels within each division and by the network response time between divisions



When you are benchmarking the interactive performance of computers executing a transaction program (TP) on the host, several response time and resource figures are of primary value. These include the host internal response time, external computer response time, network delays, and the true user response time, CPU use, and transaction rates.

In This Section

Host Internal Response Time

External Computer Response Time

Network Delay

True User Response Time

See Also

Other Resources

Transaction Integrator Performance Guide

Host Internal Response Time

The host internal response time is the time that the transactions spend inside the host system processing the transaction requests from Transaction Integrator (TI). These include components such as processing the business logic, disk and database I/O, and handling the two-phase commit (2PC) processing.

One way to get an indication of the host internal transaction processing performance is to run the transactions under load without the TI and computer components. A typical goal for an interactive transaction is to keep the response time at 500 ms and less than one second for most part of the daily load conditions. This is a tough goal, and many systems under ordinary operational loads have hard times maintaining this. There is a section dedicated to analyzing the effects on the application server in case the response times get longer.

See Also

Other Resources

Major Elements Affecting Overall Performance

External Computer Response Time

External computer response time is sometimes referred to as end-user response time or end-to-end response time. It is the actual response time that the client application that is running on the computer sees.

When you are using the "FAT-client" architecture, it includes all processing and network delays incurred to the transaction from the time DCOM call was sent to the TI server until the time the reply for the transaction comes back. For "thin client" application architecture, additional delays are incurred because Internet Information Services (IIS) converts the ASP requests to DCOM calls, and delivers the DCOM replies back to the ASP pages for the client on the intranet or Internet.

See Also

Other Resources

Major Elements Affecting Overall Performance

Network Delay

Network delay is the difference between what the host measures as the internal transaction response time and what the client computer sees as external response time, excluding the processing time on the application server. On a high-speed local area network (LAN) deployment, the contribution from the network delay can be very small, but when wide area networks (WAN), satellite hops, or modems are involved, they can be major contributors to poor end-user response times.

See Also

Other Resources

Major Elements Affecting Overall Performance

True User Response Time

The true user response time is the time that the whole transaction takes to process. This is measured on the user-interface level. The difference in the true user response time and the external computer response time for Transaction Integrator (TI) transactions depends on how much of the processing is done on the client itself. For the FAT client approach, the opportunities to have "business logic" on the client side are greater than that of the thin client. The thin client processing typically involves just screen presentation processing delays.

The response time for FAT client transactions through TI, when the host processing time is virtually zero, is at maximum approximately 50 milliseconds for a small transaction (481 KB in/out). This is measured by the VCperform client application, and represents very closely the true end-user response time, missing only the screen presentation processing time. The amount of data conversion, heavy or light, and also using selection hints and UDTs did not affect the response time.

This response time includes the LAN delays for both TI processing and the backend host simulation processing; it is as close as possible to the optimum possible performance.

Response Time Contributors

On a properly tuned system, TI processing generally contributes less than 50 ms to the overall user response time. Two-phase commit (2PC) adds approximately 100 ms to this as a result of the disk I/O for the Microsoft Distributed Transaction Coordinator (DTC) logging.

The most significant contributor to the overall response time is naturally the host, where most of the work is done (business logic and database access). So the area to focus first in optimizing the performance is the host. To get a better idea of the response time and transaction volumes, use the TI performance counters.

See Also

Other Resources

Major Elements Affecting Overall Performance

Performance Monitoring Counters

Transaction Integrator (TI) has 24 basic performance monitoring counters that you can add to the Windows System Monitor to analyze performance and find out where the bottlenecks are in your system. You can select any of the counters, and then click **Explain** to get information about that counter.

The 24 TI performance monitoring counters are as follows:

- **Active Clients**

  Displays the total number of active clients, which are those clients that have created an instance of a TI object but have not yet released that instance.

- **Average method call time**

  Measures the average number of seconds of elapsed time that TI uses to process method calls made by the client application. The time begins when TI receives the request from the client application (the **Invoke** call). The time ends when TI returns control to the client application. This counter includes the host response time, and it is not specific to any TI programming model. Consider the following two facts when you are using this counter:

  - Special TI properties, such as **GetNewRecordsSet**, have been omitted from the calculation.
  - Two-phase commit (2PC) response time is not considered and is omitted from the calculation.

Note
Host response times for CICS, CICS Non-LINK, and IMS can be subtracted from the average method call time to calculate the amount of time TI spends processing methods. For example, assume a transaction takes one minute to complete. The host response time is 48 seconds and the average method call time is 60 seconds. Subtracting the host response time from the average method call time leaves 12 seconds that TI uses to process the methods. The host uses most of the transaction time.

- **Bytes recv'd from a TCP host / sec**

  Displays the number of bytes per second received from the mainframe by TI over the TCP/IP protocol. This counter is not specific to any TI TCP/IP programming model. For the CICS MSLink model, the number reported will be slightly more than the amount of user data due to the Link model protocol header data. Bytes received from the host represent all data traffic, including user data.

- **Bytes recv'd from an SNA host / sec**

  Displays the number of bytes per second received from the mainframe by TI over the SNA protocol (APPC/LU 6.2). This counter is not specific to any TI SNA (APPC/LU 6.2) programming model. For the CICS Link model, the number reported will be slightly more than the amount of user data due to Link model protocol header data. Bytes received from the host represents all data traffic, including user data and two-phase commit (2PC) flows.

- **Bytes sent to a TCP host / sec**

  Displays the number of bytes per second sent from TI to the mainframe over the TCP/IP protocol. This counter is not specific to any TI TCP programming model. For the CICS MS Link model, the number reported will be slightly more than the amount of user data due to Link model protocol header data. Bytes sent to the host represent all data traffic over TCP/IP, including user data.

Note

To determine the TI load on the Host Integration Server computer, you can compare the average number of bytes sent and received by TI with the corresponding performance counters for Host Integration Server. For example, if the averag e number of bytes sent to the host from TI is 20 and the average number of bytes sent to the host by Host Integration S erver is 100, Host Integration Server traffic is responsible for most of the load. Consequently, the amount of informatio n coming back from the host could be less than that going to the host. That is why two counters are available, one for t he number of bytes sent and one for the number of bytes received.

- **Bytes sent to an SNA host / sec**

  Displays the number of bytes per second sent from TI to the mainframe over the SNA (APPC/LU 6.2) protocol. This counter is not specific to any TI SNA (APPC/LU 6.2) programming model. For the CICS Link model, the number reported will be slightly more than the amount of user data due to Link model protocol header data. This number is represented in terms of bytes per second.

- **Calls currently executing**

  Displays the number of method calls that are currently being executed.

- **Cumulative calls**

  Displays the total number of method calls that have occurred since the COM+ application was started.

- **Host resp time CICS Link**

  Measures the average time the host spends processing the transaction program's unit of work when the CICS Link model is in use. In other words, this counter measures how long the host takes to respond to a request. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time. Consider the following two facts when you are using this counter:

  - Two-phase commit (2PC) response time is not considered and is omitted from the calculation.
  - Multiple receives might be contained in the first response buffer sent to TI. Therefore, the response time ends when TI has obtained all receives to the first response buffer.

- **Host resp time CICS Non-link or IMS**

  Measures the average time the host spends processing the transaction program's unit of work when either the CICS Non-LINK or IMS models is in use. In other words, this counter measures how long the host takes to respond to a request. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time. Consider the following two facts when you are using this counter:

  - Two-phase commit (2PC) response time is not considered and is omitted from the calculation.
  - Multiple receives might be contained in the first response buffer sent to TI. Therefore, the response time ends when TI has obtained all receives to the first response buffer.

- **Host resp time TCP Concurrent Server**

  Measures the average time the host spends processing the transaction programs unit of work when the TCP/IP CICS Concurrent Server model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time.

- **Host resp time TCP Explicit**

Measures the average time the host spends processing the transaction programs unit of work when the TCP/IP IMS Explicit model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time.

- **Host resp time TCP Implicit**

  Measures the average time the host spends processing the transaction programs unit of work when the TCP/IP IMS Implicit model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time.

- **Host resp time TCP MS Link**

  Measures the average time the host spends processing the transaction programs unit of work when the TCP/IP CICS MSLink model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time.

- **Host resp time TCP IMS Connect or OTMA**

  Measures the average time the host spends processing the transaction programs unit of work when the TCP/IP IMS Explicit model is being used. This average time counter measures the time the host takes to respond to a request sent to it. The time starts after TI sends the final data buffer and ends when the first response buffer is received by TI. This counter is represented in terms of seconds of elapsed time. IMS Connect or OTMA enables customers to connect to existing IMS transactions without linking listeners to the transaction programs (TP), so you do not have to recompile your IMS TP.

- **Link calls / sec**

  Displays the number of method calls that use the CICS LINK programming model. This number represents of calls per second.

- **Non-link calls / sec**

  Displays the number of method calls that use the CICS Non-LINK or IMS programming model. This number represents calls per second.

- **TCP Concurrent Server calls / sec**

  Displays the number of method calls that use the TCP/IP CICS Concurrent Server programming model. This number represents calls per second.

- **TCP Explicit calls / sec**

  Displays the number of method calls that use the TCP/IP IMS Explicit programming model. This number represents calls per second.

- **TCP Implicit calls / sec**

  Displays the number of method calls that use the TCP/IP IMS Implicit programming model. This number represents calls per second.

- **TCP MSLink calls / sec**

  Displays the number of method calls per second that use the TCP/IP CICS MS Link programming model.

- **TCP OTMA calls / sec**

Displays the number of method calls that use the TCP/IP OTMA programming model. This number represents calls per second.

- **Total calls / sec**

  Displays the total number of method calls per second that TI has processed. This counter is not specific to any TI programming model.

- **Total errors / sec**

  Displays the total number of method calls per second that have returned a non-zero HRESULT to the client application. This counter is not specific to any TI programming model.

Note
Comparing method call errors with CICS LINK calls and CICS Non-LINK calls shows the severity of a given situation. For example, if the Method call errors counter reports two errors per second and LINK calls or CICS Non-LINK calls are reporting 50 method calls per second, it may indicate that one client application has trouble with a specific host application. Consequently, if the Method call errors counter reports 50 errors per second and CICS LINK calls or CICS Non-LINK calls are reporting 50 method calls per second, this indicates that a connection to the host might have been terminated.

In This Section

How to Add TI Performance Counters to Windows 2000 System Monitor

Method Calls per Second

Average Method Call Time

Errors Per Second

Host Response Time

Bytes Sent Per Second

Bytes Received Per Second

How to Add TI Performance Counters to Windows 2000 System Monitor

Follow these steps to add Transaction Integrator (TI) performance counters to Windows 2000 System Monitor.

To add TI performance counters to Windows 2000 System Monitor

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Performance**.
2. Click **+** at the top of the details pane.
3. In the **Add Counters** dialog box, move to the top of the **Performance object** box, and then click **COM Transaction Integrator**.
4. Select all the TI performance counters that you want from **Select counters from list**, and then click **Add**.
5. Click **Close** to close the **Add Counters** dialog box.

See Also

Other Resources

Performance Monitoring Counters

Method Calls per Second

The counter reports the method call volume going through the Transaction Integrator (TI) server. There are actually three counters implemented:

- Method calls using the CICS LINK mode.
- Method calls using the CICS non-LINK or calls to IMS.
- Total method calls.

Assuming that the system is in somewhat stable condition, that is, the calls are returning at the same rate that they are made, these counters represent the transactions per second throughput number for TI.

See Also

Other Resources

Performance Monitoring Counters

Average Method Call Time

The average method call time performance counter represents the response time the transaction has from the time Transaction Integrator (TI) receives the method call, until the time it sends the reply back to the client. This does not include the LAN/WAN delays between the client and TI server.

See Also

Other Resources

Performance Monitoring Counters

Errors Per Second

The errors per second performance counter indicates whether the method calls are failing on errors. Under normal operation, this value should remain at zero or very close to zero.

See Also

Other Resources

Performance Monitoring Counters

Host Response Time

The host response time indicates the response time observed by the Transaction Integrator (TI) transport component, measuring the time from the call to the server computer, or TCP/IP stack, until the reply from the host. This includes some networking overhead. In a typical well-tuned high bandwidth LAN environment, this response time should be very close to the actual host processing time.

You should consider some important tuning issues, especially with SNA Link, in order to declare this to be the representative figure for the host. For more information, see SNA Link Tuning. For the host response time, TI again separates the CICS LINK mode, and the CICS non-LINK or IMS mode into two separate counters.

See Also

Other Resources

Performance Monitoring Counters

Bytes Sent Per Second

The bytes sent per second performance counter reports the total data flow in bytes from the Transaction Integrator (TI) server to the host per second. This becomes a useful counter when TI transactions become large, and communication links can become the bottleneck in the system. This is particularly important when WANs are used.

See Also

Other Resources

Performance Monitoring Counters

Bytes Received Per Second

The bytes received per second counter reports the total bytes received from the host per second. The bytes received together with the bytes sent represent the total data transfer on the communications links between the Transaction Integrator (TI) server and the host computer.

See Also

Other Resources

Performance Monitoring Counters

Windows Server Tuning

Use the topics in this section to help reach optimum operating system performance for Windows 2000 Server.

In This Section

Adjusting Application Priority

Reducing Context Switching

Streamlining Authentication

How to Optimize Network Throughput in Windows

See Also

Other Resources

Transaction Integrator Performance Guide

Adjusting Application Priority

Under ordinary loads, there is no need to adjust the Windows tasking priorities. However, you can increase server task performance out of the system to cover peak loads by adjusting the balance between background and foreground applications. An even balance between the two gives background applications a better response time, but still gives more processor time to the foreground application.

To adjust the balance between background and foreground applications in Windows 2000

1. Open **System** in **Control Panel**.
2. Click the **Advanced** tab.
3. Click **Performance Options**, click **Background services** under **Optimize performance for**, and then click **OK**.

See Also

Other Resources

Windows Server Tuning

Reducing Context Switching

Context switching reduces server performance on any operating system: Windows 2000 is no exception. If a system is doing 50,000+ context switches per second (unlikely, but possible), it does not have time to do actual work. Instead, it is spending all its time switching various code and data pages in and out of its memory to L2 cache, RAM, or even to the disk drive; in other words, the system is *thrashing*.

Adjusting Thread Count

You can reduce the amount of context switching by reducing the total number of active threads. The topics in the Transaction Programs that Run for a Long Time section describe two registry entries that can increase the thread counts for a COM+ application and Transaction Integrator (TI) two-phase commit (2PC) transactions. Be sure to adjust the thread counts to gain optimum performance. Always track the context switching per second when making adjustments to these two registry entries.

In addition to adjusting the COM+ and TI threading, you can also adjust the Host Integration Server asynchronous I/O threading model. These threads serve the Host Integration Server client traffic only, and are designed to handle the full 30,000 sessions hitting Host Integration Server with over 1,000 transactions per second (TPS). Performance testing in the Microsoft lab environment has shown no reason for adjusting these values, but if you need to minimize the thread count in your system, you can make adjustments because TI represents only one client to Host Integration Server.

Use the following information to calculate the thread count:

- The Host Integration Server I/O thread count is equal to the base thread count plus the additional threads per CPU multiplied by the number of CPUs.
- The base thread count is five.
- By default, there are an additional four threads per CPU.

To adjust the thread count, specify the number of additional threads per CPU by adding a DWORD value *NumberofIOthreads* to the \parameter registry location. You can have up to 64 I/O threads per system.

See Also

Other Resources

Windows Server Tuning

Streamlining Authentication

Transaction Integrator (TI) users are typically authenticated for both the Windows domain and the host domain on a transaction-by-transaction basis. Host Integration Server provides the necessary security integration between these systems.

You can set the *Already verified* parameter to streamline the authentication on the host side. Both TI and the Host Integration Server node maintain a cache of verified domain/user IDs in a secured location. To guarantee fast access to the Windows authentication, install the primary or backup Windows 2000 domain controllers, TI, and Host Integration Server all in the same LAN segment. Installing these helps to eliminate delays caused by bridging or routing.

See Also

Concepts

Transaction Integrator User's Guide

Other Resources

Windows Server Tuning

How to Optimize Network Throughput in Windows

For Windows 2000 and Windows Server 2003, you can adjust networking priority to guarantee the best possible networking throughput.

To optimize network throughput in Windows

1. Click **Start**, point to **Settings**, and then click **Network and Dial-up Connections**.
2. Right-click a connection, and then click **Properties**.
3. Do one of the following:
   - If this is a local area connection, on the **General** tab, in **Components checked are used by this connection**, click **File and Printer Sharing for Microsoft Networks**, and then click **Properties**.
   - If this is a dial-up, VPN, or incoming connection, on the **Networking** tab, in **Components checked are used by this connection**, click **File and Printer Sharing for Microsoft Networks**, and then click **Properties**.
4. Click **Maximize data throughput for network applications**.

See Also

Other Resources

Windows Server Tuning

SNA Communication Tuning

The topics in this section describe the major factors that affect Transaction Integrator (TI) throughput over the APPC/LU 6.2 SNA transport.

In This Section

LU 6.2 Contention Winner Limit

Pre-Activation of the LU 6.2 Sessions

SNA Link Tuning

Host (VTAM, CICS or IMS) Response Time

See Also

Other Resources

Transaction Integrator Performance Guide

LU 6.2 Contention Winner Limit

Transaction Integrator (TI) uses only LU 6.2 contention winner sessions. The number of sessions negotiated with the host system determines the number of sessions available for concurrent TI client requests. The parallel session limit and contention winner limit should be set to the same value within the Host Integration Server APPC mode definition used by TI.

Note
Other APPC applications can share the same Local APPC LU, Remote APPC LU, and Mode used by TI. If they do, you must define sufficient sessions to handle the needs of all applications.

In addition, the Request/Response Unit (RU) size should be sufficiently large to hold the standard message size sent between TI and the host application. For example, if the host response is expected to exceed 2 KB, set the Max Send and Receive RU size to 4096 within the Host Integration Server APPC mode definition. The session level pacing values will not likely affect transaction performance because a transaction may only involve a single request and single (under 4 KB) response. In this case, an APPC mode send and receive window of two or more should be sufficient. However, if a large host response is expected, it is recommended to use a larger RU size to reduce SNA-level message acknowledgements.

Troubleshooting Suggestions

Before starting a connection, enable Host Integration Server data link control (DLC) and LU 6.2 message traces of the connection startup and initial TI component use. Provide the traces to an SNA support engineer. This engineer can decode the DLC trace to determine the parallel session limit and contention winner limits negotiated in the Change Number of Sessions (CNOS) exchange for the LU/LU/mode used by TI.

See Also

Other Resources

SNA Communication Tuning

Transaction Integrator Performance Guide

Pre-Activation of the LU 6.2 Sessions

Activating LU 6.2 sessions in advance will automatically prevent a short delay in establishing new LU 6.2 sessions as Transaction Integrator (TI) conversation allocation requests are made. In addition, pre-activating sessions keeps idle sessions active during the SESTIMEOUT period (approximately 20 seconds).

Host Integration Server honors its APPC mode automatic activation limit setting for any APPC LU/LU partnership defined within the APPC mode when the connection is initially activated. Partnerships are defined in the **APPC-mode Partners** tab.

For more information about the *SesTimeout* registry parameter, search the Host Integration Server online Help.

See Also

Other Resources

SNA Communication Tuning

SNA Link Tuning

If you are using token-ring, Ethernet, or FDDI to communicate with your host system, investigate data link control (DLC) tuning. The following 802.2 connection default settings should be sufficient:

- Unacknowledged Send limit is 8 by default.

- Receive ACK threshold is 2 by default. However, some hosts are set to a Send Window of 1, so they require an ACK from Host Integration Server for every frame they send; whereas with the default of 2, the host can stop sending until the ACK has been received. The DLC will then go through the time-out and recovery, but the performance will be decreased significantly. It is important that you set each node's send window to be larger than its partner's receive window.

- Maximum BTU length: 1929 for token-ring, 1493 for Ethernet. If a 16-Mbps token-ring is being used, and the TI request or host response will exceed this BTU length, increase the maximum BTU length to 4105 (or 8192 for the maximum possible) within the Host Integration Server connection.

Troubleshooting Suggestions

Capture a Network Monitor or Sniffer trace of Host Integration Server computer-to-host traffic, and provide it to an SNA support engineer. The engineer can use this trace to observe the Send Window and Receive ACK threshold being used by both ends by looking at the LLC traffic.

See Also

Other Resources

SNA Communication Tuning

Host (VTAM, CICS or IMS) Response Time

The host response time, also called the unit of work (UOW) or host processing time, for each transaction affects the number of transactions that can be performed given the number of LU 6.2 sessions that are used.

If CICS is used, investigate the CICS region definitions for parallel session limit and contention winner limit. These values are configured in the Maximum parameter in the CICS region SESSION PROPERTIES:

SESSION PROPERTIES Maximum==>100, 000

The first value is the parallel session limit and the second value is the CICS contention winner limit. For Transaction Integrator (TI) use, Host Integration Server should be configured as the contention winner for all sessions, so you should set the CICS contention winner limit to zero (0). Also, verify that the CICS region maximum tasks value is sufficient to handle the concurrent client requests.

If you are using IMS, verify that IMS has sufficient message processing regions to handle the expected load.

Troubleshooting Suggestions

Capture a Host Integration Server data link control (DLC) message trace of the throughput test and analyze the host response time observed on the LU 6.2 sessions.

Within a Host Integration Server DLC message trace, a unique LU 6.2 session is distinguished by a unique Originating Address Field (OAF), Destination Address Field (DAF), and OAF/DAF Assignor Indicator (ODAI) values. The OAF and DAF specified in a Host Integration Server session request will alternate on the host response.

Note
Either end can deallocate the conversation, although this is often done by TI. However, it is possible for the host data respons e to contain the Conditional End Bracket (CEB).

See Also

Other Resources

SNA Communication Tuning

SNA vs. TCP/IP

TCP/IP is not as scalable as SNA with Transaction Integrator (TI), but TCP/IP is more effective in other areas such as file transfer and data access (about 10-15% better on OLEDB tests). In addition, the IBM endorsement of TCP/IP simplifies the corporate networking infrastructure and enables easier interoperability to the Internet.

To study the performance differences for TI with TCP/IP as the up-stream link protocol versus SNA, Microsoft performed a stress test where the only change to the setup was the up-link protocol. The SNA test was set to use the **SelectionHint** property setting because this is the same process performed by the TCP/IP up-link. For more information about the effects of the **SelectionHint** property, see Remote Environment Selection Using the SelectionHint Property.

Analyzing the average response time as a function of transactions performed, Microsoft found that TCP/IP is as fast as SNA. In fact, TCP/IP is faster than SNA when the load is below 100 transactions per second (tps). For some deployments, that is the typical operating range. As the load increases, the connectionless TCP/IP up-link starts to slow down the response time and clips the top performance at 500 tps. The SNA up-link enables stable response times throughout the load range, and therefore achieves higher scalability.

To analyze the superior scalability of SNA compared to TCP/IP, Microsoft charted the frames per second over the backbone LAN. Whereas SNA maintains its sessions from transaction to transaction, TCP/IP must establish and destroy the TCP/IP connection for each transaction. This results in more frames transmitted over the up-link compared to SNA. In our tests, the 100baseT Ethernet LAN did not become a bottleneck, but this can become the critical issue if the link speed between the TI server and the host is slower. In any case, the TCP/IP connections and disconnections generate some additional interrupts for both ends.

Additional analysis shows an advantage that the TCP/IP up-link has over the SNA up-link that might become the deciding factor in a real-world deployment. When you are using the SNA up-link, the TI Automation server must endure almost twice the amount of context switching compared to the TCP/IP up-link case. This is because with SNA, the messages are passed from TI first to the SNA server node and then to the SNA link service. These are separate processes, so there is process-to-process communication and context switching. With TCP/IP this does not occur; TI passes the messages directly to the NDIS driver. On a server where additional processing of business logic is occurring, so much context switching may occur that throughput and scalability are affected.

See Also

Other Resources

Transaction Integrator Performance Guide

System Sizing

System sizing affects performance. By using the tips in this section you can achieve optimal system sizing.

In This Section

- LAN Throughput
- Escon Channel Throughput
- Windows 2000 Services

See Also

Other Resources

Transaction Integrator Performance Guide

LAN Throughput

The **100baseT** alternative to connect the Transaction Integrator (TI)/Host Integration Server computer to your host system is the most popular and currently the most available. This topic explains the bandwidth offered by **100baseT**, and the amount of that bandwidth that you can actually start using productively.

Calculating the Maximum for 100baseT

You can calculate the theoretical maximum in the following way for the **100baseT** Ethernet:

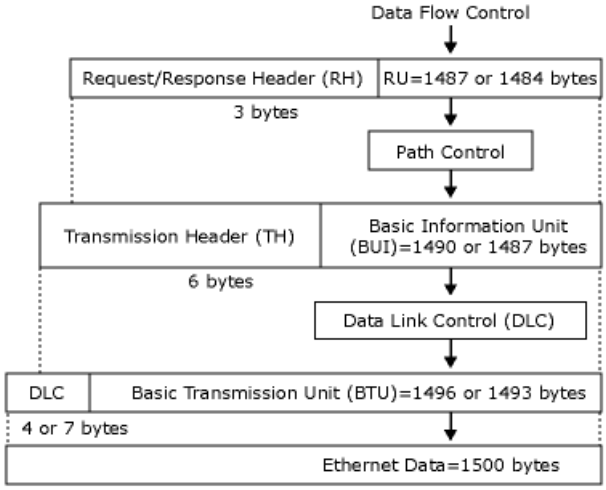
100BASE-T is clocked at 100 MHz, with a 25-MHz crystal multiplied by 4. The coding is 8/10, meaning one byte is packaged into 10 bits. Therefore, you can at most transfer 100/10=10 million bytes per second. To convert this number to megabytes per second (MBps), divide it in the following way:

*10,000,000/(1024*1024)=9.5 MBps*

Then there is the question of efficiency. Ethernet provides up to 90-95% efficiency (CSMA-CD). There is a maximum payload of about 1500 bytes per frame, and some minimum inter-frame spacing. Also, if you use half-duplex cabling, the ACK packets must take the bus sooner or later, making it almost impossible to reach the maximum.

The frame format for 802.2 over Ethernet is at maximum 1487 bytes, or 1484 depending on the Ethernet standard used IEEE, or DIX. The following figure shows the maximum RU and BTU sizes over an Ethernet.

Data flow control showing maximum sizes over an Ethernet: 1487/1484 bytes for RU, 1490/1487 bytes for BIU, 1496/1493 for BTU, and 1500 bytes for Ethernet data



The format for TCP/IP over Ethernet is 14-byte Ethernet layer+20 IP+20 TCP+12 (TCP-timestamp)+1448 data. For each packet, the header overhead is 54/66 bytes. Of course, there are the ACK packets, one every two packets in TCP/IP. Therefore, the header overhead is three headers for two data packets, which is around 7-8%.

For 802.2 data link control (DLC) traffic, the acknowledged frequency is controlled by each end negotiating with its partner. For more information, see SNA Communication Tuning.

For the 90–95% efficiency referred to previously, the throughput is affected by various other factors, such as the size of the broadcast domain, whether the LAN is on a switch or a hub, the number of servers sharing the segment causing possible collisions, and whether your network has other protocols such as IPX, whose broadcasts can consume some of the available bandwidth.

Looking at the LAN usage levels in lab tests on an isolated switched 100baseT, with only a few servers on the segment, we should be getting close to the theoretical maximum minus the known overhead. Can TI push the LAN to the maximum performance?

The test results show that when sending 32000 bytes, and receiving 32,001 bytes back, TI can drive the 100baseT close to its maximum performance if there is only minimal data conversion and no other "business logic" or processing competing with TI on the server. This is, of course, with an isolated optimized network. The backbone network in the real world must endure a lot more overhead without becoming the bottleneck for the system. To be on the safe side, a prudent design criterion for 100baseT LAN would be to keep the planned load as follows:

- Less than 4 MBps for systems that mainly move data.
- Less then 3 MBps for systems with short interactive transaction messages.

The reason for designing the interactive LAN load to a lower limit is due to the higher number of frames per MBps. Observing these criteria will set the peak LAN load to a safe 50% of the LAN's capacity.

See Also

Other Resources

System Sizing

Escon Channel Throughput

Although the 100baseT LAN can operate with 3–5 MBps throughput range on a heavily loaded system, the Escon channel specified at 17 MBps can reach very close to its maximum specified throughput. Tests done with Host Integration Server on a channel-attached quad PP200 were able to reach 12 MBps against a large mainframe using LU 6.2 doing straight memory-to-memory transfer. This rate is not typically reached except during system backup procedures or database distribution.

To place this result in the right context for a large online transaction processing (OLTP) system doing 500 TPS with transactions typically consisting of 200-byte input and 1900-byte reply, the Escon overall data rate would be 1 MBps plus some overhead. The Escon channel rarely becomes the system bottleneck because one adapter can support multiple 17-MBps channels using the Escon Multiple Image Facility (EMIF).

See Also
Other Resources
System Sizing

Windows 2000 Services

The topics in this section discuss ways in which Windows 2000 services affect your planning and deployment.

In This Section

Normal Load

Preparing for Running in Degrade Mode

Considering System Growth

Estimating the System Load

See Also

Other Resources

System Sizing

Normal Load

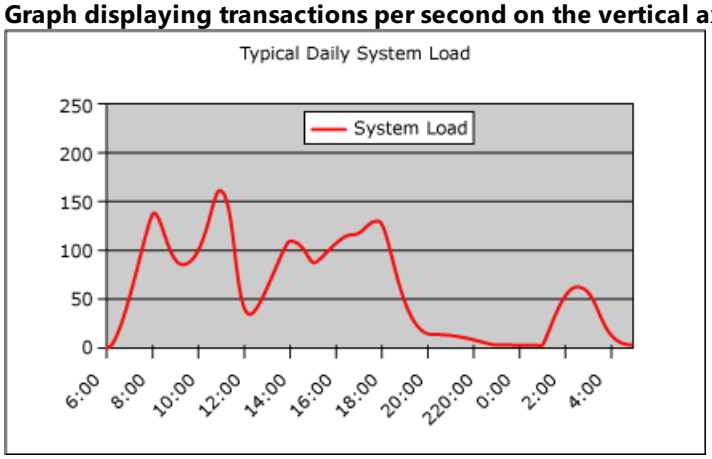
To size the Windows 2000 applications server, you must review system behavior over the course of a day, and also over the course of a longer period of time (typically matching the accounting period to capture the month-end processing).

Analyzing Load Patterns

Analyzing these load patterns gives a more accurate view of the requirements for the system sizing. The following chart illustrates a hypothetical business system load on a typical business day. If you make the effort to capture the daily load spread, Mondays or Fridays typically give the most information about the daily swings on the load. The chart shows an average workload of 60 TPS over 24 hours, and 100 TPS over 12 hours—a business day. If you used the daily averages for the system sizing, (peak loads at 8:00 AM, right before lunch, and close of business at 5:00 PM), the result is system overload and poor performance. This poor performance would irritate users daily and cause them to change their work patterns, yielding less efficient operations.

The following figure shows a typical system load.

Graph displaying transactions per second on the vertical axis and hour of the day on the horizontal axis



Additionally, at less frequently occurring high peaks, such as month-end processing, the required peak load handling is increased if the processing overlaps with the daily peak loads. If the extra processing can be scheduled during low load time periods, for example, 10:00 PM to 1:00 AM, no extra capacity is needed.

See Also

Other Resources

Windows 2000 Services

Preparing for Running in Degrade Mode

Because Transaction Integrator (TI) is a combination of application server and transactional gateway between COM applications and CICS and IMS applications, some level of fault tolerance is required. On large OLTP systems, for example, serving a company's order entry, shipping, and inventory management, system uptime is an essential requirement for success. Levels of fault tolerance can be achieved as described in Load Balancing and Hot Backup.

When sizing the system to match the throughput requirements, you must provide enough processing power on the remaining servers if one server in the cluster fails. For load balancing two servers under ordinary conditions, each server should be able to handle the daily peak loads alone—not necessarily at 60% CPU load, but not exceeding the 90% limit either. For three server systems, any two should be able to handle the daily peaks at 60-90% CPU levels. If you size close to 90% CPU on a degraded operation, a slight response time impact can be expected. This is acceptable, if the recovery from the failure does not last more than a day or two. This usually means that a spare server is available locally and that it can be easily configured for service.

You should also study the excess load caused by the recovery process, its duration, and impact on the system responsiveness during that time.

See Also

Other Resources

Windows 2000 Services

Considering System Growth

Estimating system growth can be difficult. Careful study of the functional requirements, business plans, market growth, and user community might not provide the exact answers, but it can provide a rough estimate of the growth potential for the system. For a successful company, this growth rate can be as high as 25% per year, and usually during the first year from the deployment even as high as 50% per year. If our projection for the system growth follows these lines, we must deploy with 50% extra capacity at hand, either as larger servers, or as additional servers available for load balancing. Looking at a 5-year growth path, the plan can include changing to more powerful servers, adding more SMP CPUs to the servers, or adding more servers into the cluster.

The following tables show an example where a typical system with daily peaks of 160 TPS first grow at 50%, with 25% growth after that. The first table shows the server rating at various CPU% based on the applications used and tested.

Server	**TPS at 60%**	**TPS at 90%**
Dual Xeon 400	110	160
Quad Xeon 400	220	320

The following table shows the growth projections.

	First year	**Second year**	**Third year**	**Fourth year**	**Fifth year**	**Sixth year**
Growth %	50%	25%	25%	25%	25%	
Daily peak TPS	160	240	300	375	468	585
Deployment	Two dual 400s in a cluster.	Three dual 400s in a cluster.	No upgrade	One dual two quad 400s in a cluster.	No upgrade.	Three quad 400s in a cluster.
Action		Add a dual server to the cluster.		Add two CPUs to two servers.		Add two CPUs to one server.
Reason		Dual 400 maxes out at 160 TPS.		Two dual 400s max out at 320 TPS.		A dual and a quad max out at 480.TPS
Best practice	Two quad 400s in a cluster.	No upgrade.	No upgrade.	Three quad 400s in a cluster.	No upgrade.	No upgrade.

Creating a Plan for Growth

Creating a plan for future growth can be a time-consuming and difficult task. You can spend a lot of time testing and creating load scenarios for the future. Also, the continuing development of the hardware and software will affect your planning. Future computer systems are likely to outperform the currently available models with many 100% increase in dollars/throughput, thus causing the plan to be less accurate and outdated. To avoid this, a few rules of thumb can be helpful:

- Plan at least two growth scenarios with the hardware available today, with easily available performance data. Plan one scenario for aggressive growth, and one for minimal growth.
- For the initial purchase of the hardware/software, consider such factors as architecture, hardware availability, networking options.
- Revisit your plan annually, updating your projections and growth plans.

- Consider your growth plan in light of the future roles of the other computing systems in the enterprise: How is inventory control going to evolve? How are customer orders handled and which system is handling them? Are customer orders going to migrate from one system to another?

See Also

Other Resources

Windows 2000 Services

Estimating the System Load

The following table can help you collect the transactional load information you need for appropriate system sizing.

Element	Sample	Comments
Transaction name	MD481	
Input buffer size	481	Data that user/client application sends to CICS/IMS.
Data Conversion; L/M/H	m	Light= mostly char, Medium= 1/3 Char, 1/3 Float, 1/3 Packed Decimal.
Structured data; NO/UDT/ADO RS	no	UDT=user-defined data types, ADO RS= disconnected recordsets
Reply buffer size	481	Reply buffer size from CICS/IMS.
Data conversion; L/M/H	m	Heavy= No char data, mix of floats, date-times, packed dec.
Structured data; NO/UDT/ADO RS	no	
Transactional; YES/NO	no	YES= two-phase commits are used with CICS/IMS and other forks.
Security; NO/User/Pkg	no	Type of integrated security in use.
Client type; ASP/DCOM	asp	ASP= Web-based client, DCOM= client using DCOM to invoke pkg.
Package: Lib/Svr	svr	Library= inprocess COM pkg, Server=COMTI pkg is on its own process.
Number of users	3000	
Total average daily load; TPS	30	
Total daily peak load; TPS	70	
Minimum Host Unit of Work time in ms	350	Milliseconds spent from processing input to sending user reply.
Minimum Host prepare time in ms		Two-phase commit only.
Minimum Host commit time in ms		Two-phase commit only.
Expected user response time	800	
Maximum allowed response time	1600	Maximum acceptable response time.

Creating a System Load Spreadsheet

This table provides all the detailed information you need including the type of transactions, the load levels, and the acceptable response times. In your spreadsheet, create a "Sample" column to represent each type of transaction. For example, you might have five columns, one for each of the following transactions:

- Order header entry
- Order line item
- Inventory move
- Shipping header
- Shipment line

As you can see, on a large-scale system, with many different transactions, the number of columns can become unmanageable. In that case, you might want to flip the spreadsheet clockwise 90 degrees so that you can list the elements in the columns of the first row, and then place each transaction in a row.

Adding the columns to illustrate the amount of server, LAN, and host resources will complete the estimation. These will remain the real challenge, because each transaction and implementation is different, and cannot be standardized. Thus, all the test cases illustrated in this table are without any business logic processing included. They remain a task for the system designer to estimate.

See Also

Other Resources

Windows 2000 Services

Load Balancing and Hot Backup

Load balancing and hot backup allow for scalable applications and increased performance on enterprise systems. Deploying multiple servers increases application throughput by distributing the load based on rules that are defined by the load balancing engine. Load balancing services also increase availability by detecting connection failures and providing redundant resources for client applications.

Transaction Integrator (TI) can be deployed taking advantage of several load balancing and hot backup solutions. The topics in this section describe the various load balancing and hot backup methods that TI supports today.

In This Section

Host Integration Server Load Balancing

Transaction Integrator (TI) can use Host Integration Server load balancing and hot backup capability by deploying multiple Host Integration Server End-User Client and Host Integration Server Server computers in a single subdomain. Redundant APPC session pairs can be configured across multiple Host Integration Server computers to provide load balancing and hot backup. When a communication failure occurs, hot backup reroutes sessions to other host connections. For information about how to set up a hot backup system for two-phase commit and TI, see Providing a Fail-Safe Environment for ACID Transactions.

Autoactivating Sessions

For sessions to be spread across several servers, you must configure the mode definition to autoactivate sessions. When an APPC application (such as TI) requests a conversation, the APPC library sends a non-forced open LU 6.2 request to every node (SNA Server), which has the required Local logical unit (LU) (or a Local LU in the default pool if no LU name is specified). The node returns an error that indicates the best connection to use. The APPC library then chooses the response that has the lowest error number and issues a forced open LU 6.2 request.

LU 6.2 Errors

The errors for LU 6.2 are as follows:

0804 = Connection is disabled.

0604 = Session limits reached for LU/LU/mode.

0404 = Dependent LU - Connection active, but no LU-SSCP session active.

0204 = Dependent LU - LU-SSCP active, and PLU-SLU session already in-use.

0008 = Connection is pending.

0004 = Connection is inactive, no LU-SSCP session active.

0003 = If dependent LU, no LU-SSCP or PLU-SLU session active. If independent LU, CNOS not done yet for this LU/LU/mode.

0002 = Independent LU - CNOS done but no sessions currently active.

If the connection has an active session available (in other words, it is a bound session without a conversation established), the non-forced open LU 6.2 is processed by the node and returns a positive response to the APPC library (assuming it was successful in its request to the host).

For load balancing to work correctly, all connections must have active sessions available. If this is not the case, the first connection to establish a conversation is always chosen by the APPC library because it will return a lower error than the other connections. You can configure connections to autoactivate sessions by setting the autoactivation limit and LU partnering in the mode definition.

Configuring TI and Host Integration Server for Load Balancing

TI must also be installed on its own server independent of the two Host Integration Servers that have connections to the host. If TI is installed on either of the two servers that have connections to the host, load balancing will not function.

The Host Integration Server client process (the SnaBase service on Windows 2000) opens a sponsor connection to the SnaBase service on a Host Integration Server computer in the subdomain. This sponsor connection remains active while the Host Integration Server client process is running. When the Host Integration Server client process first starts, the client receives a list of all Host Integration Server computers in the subdomain. After that, only server changes are sent.

Host Integration Server

To configure Host Integration Server for APPC Load Balancing, define redundant local LU and remote LU aliases across Host Integration Server computers by using SNA Manager. For example:

Server 1

- Local APPC LU alias=COMTI
- Local APPC LU network name=APPN and LU name=SERVER1
- Select the **Member of default outgoing Local APPC LU pool** check box
- Remote APPC LU alias=CICS

- Remote APPC LU network name=APPN and LU name=CICS

Server

- Local APPC LU alias=COMTI
- Local APPC LU network name=APPN and LU name=SERVER2
- Select the **Member of default outgoing Local APPC LU pool** check box
- Remote APPC LU alias=CICS
- Remote APPC LU network name=APPN and LU name=CICS

Server

- Local APPC LU alias=COMTI
- Local APPC LU network name=APPN and LU name=SERVER3
- Select the **Member of default outgoing Local APPC LU pool** check box
- Remote APPC LU alias=CICS
- Remote APPC LU network name=APPN and LU name=CICS

Required Parameters

The following table references the required Host Integration Server, VTAM, and CICS parameters.

Host Integration Server	VTAM	CICS
Local Node ID—First 3 Digits	IDBLK in PU definition	Not applicable
Local Node ID—Last 5 Digits	IDNUM in PU definition	Not applicable
Control Point Name	CPNAME in PU definition	Not applicable
Max BTU Length	MAXDATA in the PU	Not applicable
Local APPC LU Name	Name in LU definition	Sessions
APPC Mode	DLOGMOD in the LU definition	Mode name
Remote APPC LU Name	Not applicable	APPLID

Transaction Integrator

To configure TI to use the Host Integration Server load balancing capability, you must do the following:

- Configure the "CICS Link using LU 6.2," "CICS using LU 6.2," or "IMS using LU 6.2" remote environments for the same local LU alias and remote LU alias defined on the Host Integration Server computer.
- Create a unique Local Node ID on each Host Integration Server computer, configured for hot backup to occur across Host Integration Server computers to a single host. (LOCADDR in the VTAM definition must be set to 0 to support independent LU 6.2.)

- Define the following registry entry on the Host Integration Server End-user client:

  KEY_LOCAL_MACHINE\System\CurrentControlSet\Services\SnaBase\Parameters\Client\ ResLocFlags: REG_DWORD: 0x8001

- In mode definition, set the autoactivation limit and LU partnering limits. This configures your connections to autoactivate sessions.

See Also

Concepts

Transaction Integrator User's Guide

Other Resources

Load Balancing and Hot Backup

Web-to-Host Load Balancing

Internet Information Services (IIS) can use Windows 2000 Network Load Balancing (NLB) to provide load balancing and fail-over support for incoming HTTP requests. NLB is a TCP/IP-based load balancing solution that load balances incoming TCP/IP packets to all nodes in a cluster or to a single node in a cluster. NLB distributes the load across identical servers.

The Host Integration Server clients and Web browsers go through IIS to gain access to the Active Server Pages (ASP) that invoke the Transaction Integrator (TI) methods that call the CICS or IMS transaction program (TP).

NLB provides high availability on enterprise systems. It detects connection failures and automatically redirects requests to other nodes in the server farm. NLB also improves performance when all incoming packets are load balanced between various nodes in the server farm based on server load.

You can configure NLB to balance the load on multiple servers that use single affinity, no affinity, or Class C. No affinity distributes all incoming TCP/IP requests across any node in the NLB server farm, which can increase the number of requests that need to be redirected because there is no concept of a session state. We recommend that you use IIS to distribute HTTP requests configured for single affinity. When the server is configured for single affinity, all incoming packets using the NLB virtual IP address are locked to a specific node in the server farm. Every packet that is sent from the client using the cluster IP address will connect to that node.

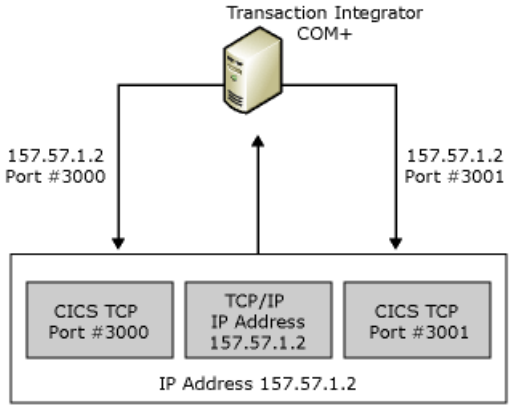
Note
NLB cannot detect whether the TI Automation server fails to respond. It can only detect if the server fails, for example if TCP/IP does not respond.

See Also
Other Resources
Load Balancing and Hot Backup

TI TCP/IP Load Balancing

Transaction Integrator (TI) can load balance TCP/IP ports when you have configured TI for CICS and IMS TCP/IP remote environments. To enable load balancing, supply multiple TCP port numbers when you create the remote environment; this enables connections to redundant CICS or IMS regions on a single host system. The following figure shows the TI TCP port load balancing solution.

Transaction Integrator receiving a TCP/IP address and sending it to CICS ports 3000 and 3001



The first TCP port is used by the first transaction and will take turns going through all the configured ports for each transaction that is invoked.

Additional host configuration details for CICS and IMS are included in the following sections.

CICS TCP/IP Platform Requirements

The version dependencies for CICS include the following:

- TCP/IP version 3R2
- CICS version 3.3 or later

Connections to CICS Using TCP/IP

CICS uses the IBM-supplied Concurrent Listener (program EZACIC02, transaction ID CSKL) to establish an interaction with TCP/IP. The Listener runs as a CICS task to help facilitate the connection process. The Listener transaction starts automatically when CICS TCP/IP is started and enabled. When the Listener starts, it obtains a socket on which it can listen for connection requests from TCP/IP. The Listener binds this socket to a specified port, and then it waits for a client request on that port. TCP/IP maintains a relationship of a port number to a CICS job. When a client makes a request on a port associated with CICS, TCP/IP forwards the connection request to the Listener in that CICS job.

TCP/IP-to-CICS Configuration

A TCP/IP port number is associated with a CICS region in the TCP/IP profile data set (hlq.PROFILE.TCPIP). The port statement is used to define this relationship. An example of a port statement that associates port 3000 with CICS job CICSRG follows:

```
3000 TCP CICSRG
```

CICS to TCP/IP Configuration

The following sample host definition shows configuration parameters for CICS-to-TCP using the EZAC transaction. The items in bold type are CICS transactions.

```
ENTER ONE OF THE FOLLOWING

CICS            ===> yes                 Enter Yes|No
LISTENER     ===>                       Enter Yes|No
```

```
ENTER ALL FIELDS

APPLID       ===> CICSRG             APPLID of CICS System

EZAC,DEFINE,CICS
OVERTYPE TO ENTER

APPLID       ===> CICSRG            APPLID of CICS System
TCPADDR   ===> TCPIP                Name of TCP Address Space
NTASKS     ===> 020                   Number of Reusable Tasks
DPRTY       ===> 000                   DPRTY value for ATTACH
CACHMIN   ===> 015                   Minimum Refresh Time for Cache
CACHMAX   ===> 030                  Maximum Refresh Time for Cache
CACHRES   ===> 010                  Maximum number of Resolvers
ERRORTD   ===> CSMT                TD Queue for Error Messages
```

The following sample host definition shows configuration parameters for the CICS Concurrent Listener using the EZAC transaction. The items in bold type are CICS transactions.

```
EZAC,DEFINE
ENTER ONE OF THE FOLLOWING

CICS         ===>                      Enter Yes|No
LISTENER     ===> yes           Enter Yes|No


ENTER ALL FIELDS

APPLID       ===> CICSRG             APPLID of CICS System
NAME         ===> xyz                  TRANSACTION NAME OF LISTENER

EZAC,DEFINE,LISTENER
OVERTYPE TO ENTER

APPLID          ===> CICSRG             APPLID of CICS System
TRANID         ===> XYZ                   Transaction Name of Listener
PORT             ===> 03000               Port Number of Listener
IMMEDIATE    ===> YES                  Immediate Startup   Yes|No
BACKLOG      ===> 010                   Backlog Value for Listener
NUMSOCK     ===> 050                  Number of Sockets in Listener
MINMSGL      ===> 004                   Minimum Message Length
ACCTIME       ===> 060                   Timeout Value for ACCEPT
GIVTIME       ===> 030                   Timeout Value for GIVESOCKET
REATIME       ===> 000                   Timeout Value for READ
FASTRD        ===> YES                   Read Immediately    Yes|No
TRANTRN      ===> YES                  Translate TRNID     Yes|No
TRANUSR      ===> YES                  Translate User Data Yes|No
SECEXIT       ===>                          Name of Security Exit
```

IMS TCP/IP Platform Requirements

The version dependencies for IMS include the following:

- TCP/IP version 3R2
- IMS version 4 or later

Connections to IMS using TCP/IP

IMS uses a Listener to establish an interaction with TCP/IP. A Listener in an IMS Batch Message Processing (BMP) helps facilitate the connection process. When the Listener starts, it obtains a socket on which it can listen for connection requests from TCP/IP. The Listener binds this socket to a specified port, and then waits for a client request on that port.

TCP/IP maintains a relationship of a port number to an IMS Listener BMP. When a client makes a request on a port associated with IMS, TCP/IP forwards the connection request to the Listener in that BMP.

Implicit Mode

Implicit mode uses the IMS Assist Module to translate conventional IMS communication into corresponding socket calls. It is dependent on the IBM-supplied default Listener (EZAIMSLN) that runs in a BMP region.

This host server application model processes input data using the IMS message queue. The Listener places the TRANID and the input data into queue. The IMS control region schedules the transaction in a Message Processing Region. The transaction program reads the request from the queue using GU and GN commands. All response data is delivered to the client by way of the ISRT command. The IBM-supplied Assist Module delivers the data directly to the client through socket API calls.

Host applications are written using CBLADLI or CBLTDLI APIs. The Assist Module uses DBLADLI API for Implicit mode. If you want an existing IMS transaction programs (TPs) to use Implicit Mode TCP/IP, you must change to the CBLADLI API and recompile the TP.

Explicit Mode

The IMS explicit (TCP/IP) model requires installation, within IMS, of the IBM-supplied default Listener (EZAIMSLN) that runs in a BMP region. This host server application model processes data without using the IMS message queue. The Listener places only a single segment (the Transaction Initiation Message) into the message queue. The IMS control region schedules the execution of the transaction into a Message Processing Region. The transaction then communicates directly with the client through socket API calls.

All IMS host server programs must be administered to IMS as no-response transactions.

TCP/IP to IMS Configuration

A TCP/IP port number is associated with an IMS Batch Processing Region (BPR) in the TCP/IP profile data set (hlq.PROFILE.TCPIP). The port statement is used to define this relationship. An example of a port statement that associates port 3000 with IMS batch region with a job name of WNWIBPR1 is:

3000 TCP WNWIBPR1

IMS to TCP/IP Configuration

An IMS MPP is started specifying the program name IMS IBM supplied Listener program (EZAIMSLN). This Listener reads a configuration file identified by the DD statement **LSTNCFG**. This configuration data set contains one or more the following startup parameter sets (one set for each transaction defined for least one CR):

- **TCPIP** statement
- **LISTENER** statement
- **TRANSACTION** statement

The **TCPIP** statement is used to identify the job name for the TCP/IP address space that will manage connection for this listener.

The **LISTENER** statement is used to specify the port number that this Listener will be using. This statement also specifies other port-related parameters such as backlog, time out values, and so on.

The **TRANSACTION** statement defines a list of transaction that this Listener can start. In addition, this statement defines whether the implicit or explicit connection mode is used.

The Listener uses these three parameter statements to inform the Listener which TCP/IP port to use and which transactions can be accessed through TCP/IP.

Here is a sample of an IMS-to-TCP/IP host definition:

```
TCPIP       ADDRSPC=WNWTCP31
LISTENER    PORT=4000 BACKLOG=50
TRANSACTION    NAME=TRANIMPL TYPE=IMPLICIT
TRANSACTION    NAME=TRANEXPL TYPE=EXPLICIT
```

See Also

Other Resources

Load Balancing and Hot Backup

Security Implications

Transaction Integrator (TI) can provide user ID and password credentials for authentication on the mainframe. They are provided in compliance with existing IBM standards, and mainframe authentication is completed by standard IBM procedures such as Resource Access Control Facility (RACF), Top Secret, and so on. All mainframe authentication is completed in a manner transparent to the developer.

When LU 6.2 is used for connectivity, credentials are transmitted to the mainframe in an SNA LU 6.2 Function Management Header Type 5 (FMH-5) ATTACH message. For more information, refer to the IBM manual, *Systems Network Architecture Formats, Document Number GA27-3136-16, Section 11.1.5 FM Header 5: Attach (LU 6.2*).

When TCP/IP is used for connectivity, credentials are transmitted in the Transaction Request Message (TRM) sent from TI to the Listener. There are some additional coding requirements on the mainframe for TCP/IP to provide user exits for authentication. For information about CICS, refer to *IBM TCP/IP for MVS CICS TCP/IP Socket Interface Guide and Reference, Document Number SC31-7131-03, Section 6.6.3 Writing Your Own Security Link Module for the Listener*. For information about IMS, refer to *IBM TCP/IP for MVS IMS TCP/IP Application Development Guide and Reference, Document Number SC31-7186-03, Section 3.4.4 The IMS Listener Security Exit*. Prior to TCP/IP version 3R2, the CICS exit module required the name, EZACICSE. However, you can choose any name when you are using TCP/IP version 3R2. For IMS, the exit module must be named IMSLSECX.

There are three alternative sources of mainframe credentials.

- The identity of the COM+ application that contains the TI component.
- The identity of the Windows user of the TI application.
- The optional explicit security override feature of TI.

Use of the explicit override feature dissociates mainframe security from Windows 2000 security; therefore, its use is not recommended over the first two alternatives. Using either of the first two alternatives integrates mainframe security with Windows 2000 security by using Host Integration Server Host Security Integration (HSI), or Single Sign-On functionality. (You do not need to install software on the mainframe for HSI unless you also want Host Password Synchronization.)

By default, passing credentials to the mainframe for authentication is not enabled. You must activate the TI remote environment (RE) security properties by selecting the **Set security on** check box. You must click either **Authenticate with package credentials** or **Authenticate with user credentials** even if you plan to use the explicit security override feature.

To select the explicit security override, select the **Allow application override** check box. This option is the least recommended of the three. If **Allow application override** is selected but not implemented by the application, the security mechanism reverts to whichever of the other two security options you selected.

Note
Explicit security override is not the preferred method of specifying credentials for a client. If possible, you should use the Client Context USERID and PASSWORD override keywords. For more information, see the COMTIContext Keywords.

In This Section

How to Use Optional Explicit-Level Override Authentication

Level of Security

Using Host Security Integration

How to Use Already Verified Authentication

Mainframe Authentication for CICS LINREs

How to Use Optional Explicit-Level Override Authentication

Clicking the **Allow application override** check box enables applications to supply credentials at run time through a callback mechanism supplied by Transaction Integrator (TI). Using application override does not require the installation and use of Host Integration Server Host Security Integration (HSI). Instead, the client application supplies TI with a pointer to a callback object that can be used to request credentials when they are needed at run time. A utility component is provided so that customers can add their callback pointer to the context, and create new COM+ objects that inherit from the modified context. The security callback component is automatically installed.

Note
Explicit-Level Override Authentication is not the preferred method of specifying credentials for a client. If possible, you shoul d use the Client Context USERID and PASSWORD override keywords. For more information, see the COMTIContext Keywords.

To use explicit security, the client application must follow these steps:

To use explicit security

1. Create an instance of an object that implements **IHostSecurityCallback**.

   This object is created in the client application and is implemented by the developer.

2. Create an instance of the TI utility object **COMTI.HostSecurityContext**.

3. Call **SetCallbackObject** on the utility object, and pass it the **IHostSecurityCallback** pointer on the callback object.

4. Create instances of its TI component by using the **CreateInstance** method on the security utility object.

When the TI component instance created in step 4 establishes a conversation with the host, it calls the **ReturnSecurityInfo** method on the callback object. TI passes this method the name of the remote environment being contacted. The output parameters provide the logon and password as clear text.

As an additional aid to developers, TI provides the type information for the **IHostSecurityCallback** interface inside the component library for the TI security component. This enables Visual Basic developers to set a reference to this component and then use the **Implements** keyword to implement the callback class.

See Also

Other Resources

Security Implications

Level of Security

Application-level (or package-level) security and user-level security are the two preferred means of authentication because they integrate security on the mainframe with Windows 2000 security. The Transaction Integrator (TI) run-time environment obtains credentials from either the Windows identity of the COM+ application or from the identity of the client application that invoked the TI Automation server that contains the TI component.

In both cases, the facilities of Host Integration Server Host Security Integration (HSI) are required. The Windows 2000 credentials are *replicated*, *unchanged*, or *mapped* to another set of credentials specific to the mainframe. The credentials are then sent to the mainframe for authentication.

Internally, the mechanism functions as follows. For COM+ application identity credentials, TI sets the user ID and password fields in the MC_ALLOCATE verb to MS$SAME, and sets the security field to AP_PGM. This informs HSI to derive host credentials for the owner of the currently executing process.

For user credentials, TI sets the security field in the MC_ALLOCATE verb to AP_PGM OR'd with AP_PROXY, and fills in the domain and account name fields in the verb ctl block with the values it obtained from **LookupAccountSid** (in the Win32 API). This informs HSI to derive host credentials corresponding to that Windows account, regardless of the running process. In other words, the run process acts as a proxy for the real user and passes the real user's credentials.

See Also
Other Resources
Security Implications

Using Host Security Integration

When you create a Host Integration Server Host Security Domain (HSD), two Windows 2000 local user groups are created. For example, if you name the HSD **OURHOST**, the two user groups created are named **OURHOST** and **OURHOST_PROXY**. There is also a Host Account Cache that is used to maintain user credential mappings. These mappings are used by Host Integration Server Host Security Integration (HSI) to map (or replicate) the original Windows 2000 credentials to host credentials.

Host Integration Server Credential Settings

The following table shows the Host Integration Server credential settings that you must specify in a given Windows 2000 user group and in the Host Account Cache when using one of the authentication options.

TI Authentication Option	OURHOST	OURHOST_PROXY	Host Account Cache
User Credentials	User	COM+ Application*	User
COM+ Application Credentials	COM+ Application		COM+ Application
Application Override	Not applicable	Not applicable	Not applicable

*The reason that you must add the COM+ application identity in the PROXY user group is because you are allowing a process (your TI component) to run under an identity other than the identity that Windows 2000 has authenticated. This group assignment specifies that you are allowing the COM+ application to run host transaction programs on your behalf.

Note

If **Application Override** is not implemented correctly by the client application, the mechanism for supplying credentials to t he mainframe falls back to one of the other two methods.

COM+ application identity defines the Component Services computer process that is running. The process can run either as an Interactive User or as a specified Windows user account. It is typical for server processes to run as Interactive User during development; however, in production, a Windows user account is usually used.

Host Account Cache

Host Integration Server maintains the Host Account Cache. For any Windows 2000 domain user, the Host Account configuration specifies whether Windows credentials are replicated to the mainframe or mapped to other credentials to be sent to the mainframe.

See Also

Other Resources

Security Implications

How to Use Already Verified Authentication

You can set the **Use Already Verified or Persistent Verification authentication** option on the remote environment (RE) Security property page in Transaction Integrator (TI) Manager.

To view or modify the Security properties for an RE

1. Start TI Manager.
2. Right-click the RE, and then click **Properties**.
3. Click the **Security** tab, and then select the **Set security on** check box.

When you select the **Use Already Verified or Persistent Verification authentication** check box, only a user ID is sent to the mainframe; that is, no password is sent, provided the mainframe partner allows it. The mainframe relies on the assumption that this user ID has already been authenticated and does not require a password. The SNA mode on the mainframe must specify this type of authentication. For CICS applications, the mode setting is determined by the ATTACHSEC=IDENTIFY parameter of the Sessions definition used for the connection.

See Also

Other Resources

Security Implications

Mainframe Authentication for CICS LINREs

If you use a CICS LINK LU 6.2 remote environment (RE), you must use resource-level authentication.

Because of a restriction imposed by the IBM Distributed Program Link (DPL) protocol, a user ID and password transmitted from the workstation by Transaction Integrator (TI) are ignored and not used for transaction-level authentication. The target CICS region expects, under the circumstances, that authentication was completed by the TI application that initiates the IBM DPL call. (Traditionally, the application that initiates an IBM DPL call is a program in another CICS region.) Instead of using credentials from the FMH-5 ATTACH, for transaction-level authentication, the target CICS region associates the default user ID for the region with the transaction ID of the CICS task (the mirror transaction).

As a result of this behavior, any attempt to secure the mirror transaction can cause an application malfunction because of a failure to authenticate.

See Also

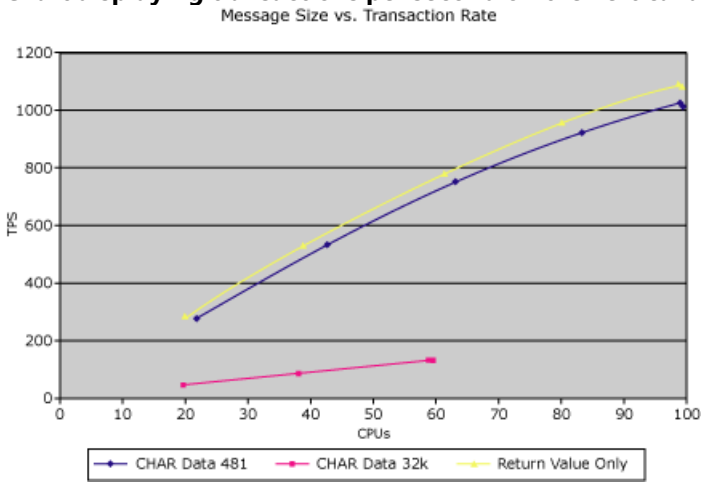
Other Resources

Security Implications

Transaction Size vs. Transaction Throughput

When looking at the transactions per second (TPS) rates that the server might be capable of sustaining, you must consider the amount of data moved and processed for each transaction. It is generally understood that the more data that you transfer for each transaction, the fewer TPS you can push through. For Transaction Integrator (TI), this also holds true. The following figure shows the rate at which the TPS decrease as transaction sizes increase on the Quad Xeon 400 test server.

Chart displaying transactions per second on the vertical axis and percent CPU on the horizontal axis



The best TPS rate is produced by the test transaction, which returns only 1 byte of data. This return value only sets the high bar to 1093 TPS. The Char481 (simple application screen) can maintain close to the maximum possible rate. The data transfer test of 32k, which moves 64000 bytes of data per transaction, cannot maintain the same TPS rate as the others. In this case, the latencies on the LAN, memory allocations, and copies begin to really show.

In This Section

Transaction Size vs. Data Throughput

See Also

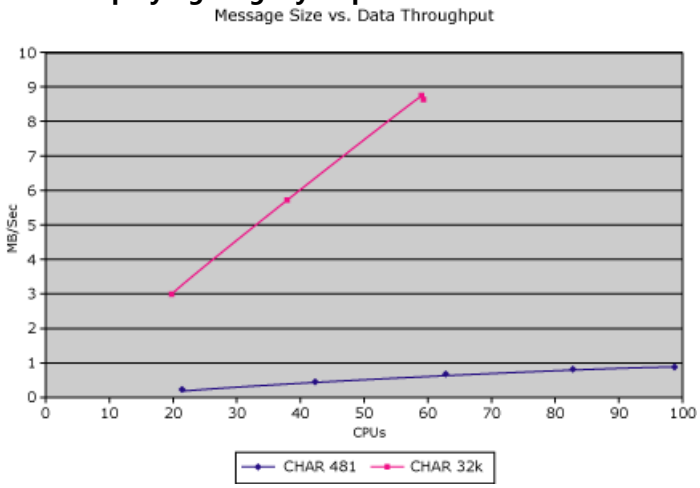
Other Resources

Transaction Integrator Performance Guide

Transaction Size vs. Data Throughput

As a result of the transaction throughput test shown in Transaction Size vs. Transaction Throughput, the question of what the best message size for these transactions arises. We captured the actual user data transferred with these transactions and looked at the MB/sec throughput versus message size. The following figure shows that if the data throughput is the objective of optimization, the larger the message size, the higher the throughput. Additional considerations can influence the selections of the message size, such as the maximum frame size on the LAN or WAN. For example, Ethernet 802.2 frames can fit 1484 bytes of user data per frame; token-ring can go up to 8186 bytes.

Chart displaying megabytes per second on the vertical axis and percent CPU on the horizontal axis



See Also

Concepts

Transaction Size vs. Transaction Throughput

Other Resources

Transaction Integrator Performance Guide

Transaction Programs that Run for a Long Time

Several special considerations apply when a transaction program (TP) runs for a long time. The topics in this section address these considerations.

In This Section

Scalability and Long-Running TPs

Processing Two-Phase Commit Transactions

Component Services User Thread Pool

TI 2PC Thread Pool

SNA Parallel Sessions

See Also

Other Resources

Transaction Integrator Performance Guide

Scalability and Long-Running TPs

When the transaction processing times (unit of work and commit times) become longer because of the nature of the transactions or an excessive load on the host, the behavior of the middle-tier software might change the throughput and end-user response time. This occurs because the number of active concurrent transactions increases and the middle-tier server is doing some level of transaction caching. For example, in a system that processes 200 TPS (transactions per second) at an average transaction response time of one second, the concurrent active transaction count at any give time is around 200. If the response time grows to five seconds, the concurrently active transactions grow to approximately 1000.

Transaction Caching

When sizing the middle-tier server, you must consider queries and batch jobs that run from seconds to hours. You can allow more transaction requests in COM+ waiting for the host responses, but excessive transaction caching might not be a good idea because it increases thread count, context switching, and memory usage. This burdens the middle-tier server and can cause performance degradation. Therefore, use COM+ or Transaction Integrator (TI) pooling capabilities cautiously; pooling does not help in the actual transaction processing, and it does not improve the response times or throughput.

In some cases, this shows real advantages. However, when transactions run against multiple host systems and databases (multi-forked transactions), the transaction caching in the middle-tier enables each transaction in the fork to start processing immediately, with the slowest fork determining the overall response time. If the transaction is not cached, all the work is blocked, until the prior transactions are completed, thus yielding lower performance. The two thread pools that are transferred when the number of active transactions grows on the middle-tier server are the COM+ user thread pool and the TI 2PC thread pool.

See Also

Other Resources

Transaction Programs that Run for a Long Time

Transaction Integrator Performance Guide

Processing Two-Phase Commit Transactions

This topic discusses what happens with the two-phase commit (2PC) transaction while it is being processed by Microsoft Distributed Transaction Coordinator (DTC) in COM+, Transaction Integrator (TI), and CICS.

The process begins when the client application invokes a method on the TI Automation server (the COM+ application that contains the TI component). COM+ then allocates a thread for the transaction from its user thread pool, begins the transaction, and passes the method's input parameters to the TI run-time environment. This thread is blocked for the transaction until the response comes back from the CICS host. This is the unit of work time, which consists mostly of time it takes the CICS application to process the transactions business logic and gain access to the database as needed (assuming that the transmission speeds keep up with the LAN speed). When the method's output parameters are sent back to COM+ from the host, the commit message is sent to DTC. The output data is stored waiting for the commit complete message from DTC. If the unit of work time grows longer, more transactions are kept active by COM+, with each transaction occupying a thread from the COM+ user thread pool.

DTC activates the prepare phase for the transactions, causing TI to allocate a thread from its 2PC thread pool and keep it blocked until the request commit message arrives from the host. After all the forks of the transactions are prepared, DTC sends a commit complete message to COM+, and COM+ then sends the method's output parameters and return values back to the calling client application and releases the thread.

This completes the transaction for the user, but the transaction monitors (DTC and CICS) still must complete the second phase of the commit, and again, a thread from the TI 2PC thread pool is allocated for each transaction doing the second phase of the commit.

See Also

Other Resources

Transaction Programs that Run for a Long Time

Transaction Integrator Performance Guide

Component Services User Thread Pool

A thread from the Component Services user thread pool is allocated to a transaction as long as the host is processing the transaction (unit of work time). This occurs for both transactional (two-phase commit transactions), and nontransactional processing. If you have slow host or communication lines, or your transactions take a long time to process, you might have to adjust a registry entry to enable more threads for this pool. The default is 100 threads per COM+ application.

See Also

Other Resources

Transaction Programs that Run for a Long Time

Transaction Integrator Performance Guide

TI 2PC Thread Pool

The Transaction Integrator (TI) two-phase commit (2PC) thread pool is different from the COM+ user thread pool. The TI 2PC thread pool is used only for 2PC transactions. The threads are precreated, and a single process interacts with Microsoft Distributed Transaction Coordinator (DTC) to handle *prepare* and *commit* transactions. This improves the performance by eliminating thread creation and destruction for every 2PC transaction.

Default Maximum Thread Settings

You do not have to worry about overburdening this pool unless large numbers of 2PC transactions are processed. Only when *prepare* or *commit* times for the transactions become very long can queuing to interact with DTC occur.

- Default maximum threads for each CPU is 20.
- Default maximum active threads for each CPU is 19.
- Default maximum total threads for each system is 80.

You can adjust the default amounts by adding a TEXT string value to the registry location:

HKLM\Software\Microsoft\Cedar\Defaults\Threads

- IOPortPoolFactor=20
- IOPortActive=19
- ThreadPoolMax=80

Rules for Specifying Values

The following rules apply for specifying values:

- All values must be greater than zero.
- IOPortPoolFactor must be >= IOPortActive + 1.
- ThreadPoolMax must be >= IOPortPoolFactor.

Caution
Allocating too many threads can cause Windows to run out of resources, and that can cause unpredictable behavior in COM + and in Windows.

See Also

Other Resources

Transaction Programs that Run for a Long Time

Transaction Integrator Performance Guide

SNA Parallel Sessions

Each active transaction allocates one parallel session to interact with the host when SNA is used instead of TCP/IP. This session is activated when the transaction sends the attach message (allocate conversation) and released after the forget message (TpEnded). When you configure too few parallel sessions for the SNA connection, the transactions that are waiting for the active ones to release can start queuing up. To avoid this queuing problem, configure enough sessions for the worst-case scenario.

Note
SNA Server 4.0 SP3 required the use of a contention winner session, but Host Integration Server does not. Host Integration Server can use any session as it becomes available.

Host Integration Server can handle large numbers of parallel sessions. The maximum is 30,000 parallel sessions for each LU-LU pair, after which another LU-LU pair must be configured. The CICS system is a bit more sensitive to the amount of parallel sessions configured. Contact your CICS systems experts for the appropriate session count.

See Also

Other Resources

Transaction Programs that Run for a Long Time

Two-Phase Commit Performance Considerations

When a Transaction Integrator (TI) component executes within a transaction, the TI run-time environment sends a message to Microsoft Distributed Transaction Coordinator (DTC) in the COM+ environment, enlisting itself on the transaction as a special type of LU 6.2 resource manager. After TI sends its data buffer to the host and receives the reply, it calls the **SetComplete** method and returns control to COM+. At this point, the client application, or other component driving TI, can perform other work also included in the same transaction. When all resource managers have made their updates and issued **SetComplete**, the transaction's creator (which can be COM+ itself for an Auto-Transaction) sends the **Commit** method to DTC. DTC sends the first-phase (**Prepare**) message to all the resource managers, including the TI run-time environment. TI generates the **Prepare PS Header** defined in the SNA Formats, and sends it to the host. It receives a **RequestCommit** in reply, which indicates that the host updates are valid and can be committed, and passes this information back to DTC. DTC collects the votes from all the resource managers, and if all prepared okay, it force-writes a Commit record to the log and sends the **Committed** message. Again, TI translates this into an **SNA PS Header**, receives the reply, and translates this back to DTC. If everything works as planned, DTC rolls back the transaction and the APPC/LU 6.2 conversation is deallocated.

Note

Neither TI nor the AP need be concerned about an APPC or CPI/C SYNCPT verb. The decision to "take a SyncPoint" is made b y the transaction creator, is expressed in the semantics of OLE transactions, and involves all participants in the transaction, no t just the TI LU 6.2 branches. The role of TI is at a lower level; TI acts as a resource manager to DTC. It translates between the COM interfaces used by DTC, and the SNA protocols understood by the host, to perform the two phases of the protocol and enable DTC to make the Commit decision between phase one and phase two.

From a performance standpoint, guaranteeing the atomicity of the host updates adds significant, unavoidable overhead. There are two additional round-trip message flows to the host for the two-phase commit (2PC), plus the Windows message flows to enlist, and the transaction logging (forced disk writes) by DTC and on the host. Transactions that do not require a great deal of business logic processing can take twice as long or more to complete when you compare it to the same transaction without 2PC.

The only time you should configure a TI component to support ACID transactions is when the associated host transaction program (TP) modifies a mission critical resource that must be kept consistent with resources on the Windows 2000 operating system. If the TP will not modify any resource for which consistency must be guaranteed, configure the TI component as **Does Not Support Transactions**, so that 2PC is not attempted. Then you are also free to use the TCP/IP protocol. The TCP/IP protocol does not support 2PC.

TI components should never be configured as **Requires a New Transaction**. This means that you are managing the transaction remotely for the host, and it would incur the overhead of creating a new transaction, enlisting on it, and performing the 2PC exchanges with the host, but the TI method would be a transaction unto itself. It is more efficient to enable CICS and IMS to manage their own transactions. Any updates on the Windows operating system would not be part of that transaction, so they would commit or roll back independently.

Note

Additional business logic processing on the same server lowers the throughput limits, stealing some of the CPU. However, th e cost can be relatively small in the scope of the overall response time budget.

See Also

Other Resources

Transaction Integrator Performance Guide

Data Conversion Cost

The following list provides advice on selecting the data types that convert most efficiently between Automation and COBOL.

- If the source and destination data types are not strictly dictated, you can decrease the amount of CPU resource consumed by Transaction Integrator (TI) by appropriately selecting the data conversions that are performed (that is, selecting the source and destination data types wisely).

- The most efficient way to pass data is to select an Automation type of **VT_BYTE** and a COBOL data type of **PIC X** untranslated. There is no conversion performed and the data is copied as is.

- The Automation type **VT_BSTR** (a UNICODE character string) converts efficiently to COBOL **PIC X**. Be aware that a **BSTR** is not the same as a C character data type; it is a Visual Basic **String**.

- The most efficient numeric data type conversions are **VT_I2** (Visual Basic **Integer** or C **short**) to COBOL **PIC S9(4) COMP**, and **VT_I4** to **PIC S9(8) COMP**.

- If the data type you want is a COBOL packed decimal, the best choice for data conversion performance is one of the Automation integer data types. If fractional parts are required (that is, a COBOL picture like **PIC S9(5)V99 COMP-3**), the best choice for the Automation type is **VT_DECIMAL** (Decimal) or **VT_CY** (Currency).

- When the COBOL data type is zoned decimal (that is, a COBOL picture similar to **PIC S9(7)V99 DISPLAY**), the same considerations as for packed decimal apply. It is slightly more work to convert Automation data types to and from zoned decimal than it is to perform the conversions to packed decimal. If the data is used in calculations on the mainframe system, it is more efficient to use packed decimal instead of zoned decimal.

- Converting floating point data types (Automation types **VT_R4** and **VT_R8**) is, in most cases, the most expensive. Converting **VT_R4** to a COBOL **COMP-1**, or **VT_R8** to a COBOL **COMP-2** (a COBOL floating point number) data type is the most efficient conversion involving floating-point numbers.

See Also

Other Resources

Transaction Integrator Performance Guide

ADO Recordsets vs. User-Defined Types in Structured Data Tests

Tests on structured data transfer show that user-defined types outperformed ADO recordsets in CPU usage, transactions per second, and response time.

Tests were conducted using six Pentium 300 MHz clients, one quad-processor Xeon p2-400 system as a gateway, and four SNA host server computers to emulate the CICS region of a mainframe. When transferring user-defined types, the clients were under less stress, and there was much better throughput overall. User-defined types were also much less stressful to the server.

The overall response times for the user-defined types are also much less because of the metadata that recordsets contain. The metadata inside the recordsets increases the size of the data marshaled back and forth across the DCOM connection. In addition to the increased size, the metadata increases processing overhead.

See Also

Other Resources

Transaction Integrator Performance Guide

Remote Environment Selection Using the SelectionHint Property

Developers can use the **SelectionHint** property to specify a remote environment (RE) programmatically.

By setting the **SelectionHint** property, applications can specify which CICS or IMS RE should be used when servicing Transaction Integrator (TI) requests. The algorithm used by an application in selecting an RE is determined by the application code. For example, a business enterprise can use separate CICS or IMS regions when handling requests from different branches. In this case, the application chooses the RE that identifies the region suitable for the current branch.

To specify an RE in an application, set the **SelectionHint** property to the name of the RE you want to use. TI Designer automatically adds the write-only **SelectionHint** property to each TI component.

Before you can use the **SelectionHint** property to specify a remote environment (RE) programmatically, the following must be in place:

- Host Integration Server must be installed on all computers running the TI run-time environment or TI Designer.
- The TI component must be assigned to an RE even though RE selection is in use. The RE assigned to the component is used when an application that has a TI component does not explicitly set the **SelectionHint** property.

To assign a TI component to an RE, use TI Manager and follow the instructions provided in the TI online Help.

In This Section

Guidelines for Using Remote Environment Selection

Consider these guidelines when you are using the **SelectionHint** property to specify a remote environment (RE) programmatically:

- Avoid hard-coding RE names into applications. Instead, load RE names from a file or database.
- Ensure that applications are structured to handle failures when they are attempting to set the **SelectionHint** property.
- Ensure that procedures for adding and configuring REs include a mechanism to update REs referenced in the application code.
- Confirm that administrative and operational tasks do not interfere with application code that uses the RE selection. Specifically, review when and how REs are deactivated and deleted.

See Also

Other Resources

Remote Environment Selection Using the SelectionHint Property

Writing Code that Specifies a Remote Environment

Applications can set the **SelectionHint** property to specify a remote environment (RE) programmatically. By specifying the RE, the application identifies the CICS or IMS region where transaction programs are carried out when the Transaction Integrator (TI) run-time environment handles calls to the TI component's methods.

The following Visual Basic code demonstrates how to set the **SelectionHint** property:

```
Dim objExample As Object
Dim Store As String
Set objExample = CreateObject("MyComponent.MyInterface")
Open "My REList.txt" for Input as #1
Line Input #1, strRE
Close #1

objExample.SelectionHint = strRE
RtrnVal = objExample.method1(parm1,  , parmN) 'Use RE named "MyRemEnvName"
```

This example shows how the application can explicitly instruct the TI run-time environment to use the RE named **MyRemEnvName** when handling the call to **method1**. In this example, **MyRemEnvName** is the first string in the file MyREList.txt. Any method calls made after **method1** that follow the **SelectionHint** assignment are handled using the original RE that was assigned to the component, not the new one. In other words, the programmatic override of the default RE does not continue past a single method call.

If an application attempts to set the **SelectionHint** property to a string that does not correspond to the name of an RE, an error is reported, and the original RE is used.

The **SelectionHint** property can be set to a deactivated RE. However, the next method call to the object will fail because a deactivated RE was selected.

The **SelectionHint** property is optional. If the **SelectionHint** property does not specify an RE, the TI run-time environment uses the original RE.

See Also

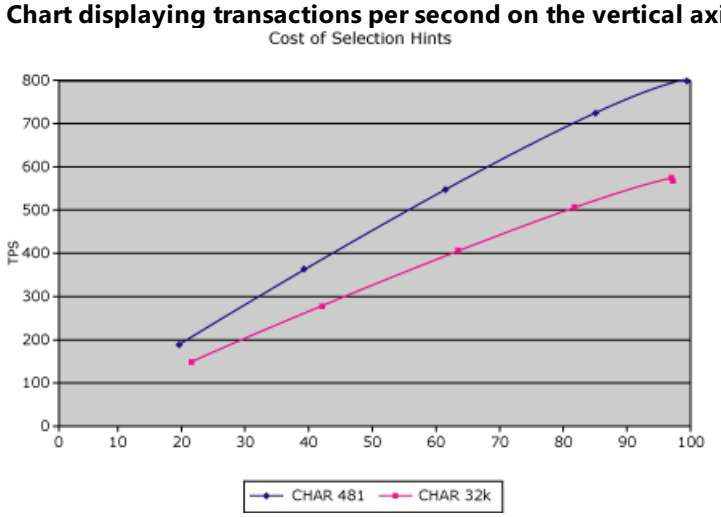
Other Resources

Remote Environment Selection Using the SelectionHint Property

Cost of Remote Environment Selection

The following figure shows the transaction throughput against the CPU load on the middle tier (the business-logic tier). Each data point represents the number of stress clients, and the total amount of time that it takes for the clients to finish their transactions. (For example, eight clients simultaneously perform transactions at a collective rate of approximately 200 TPS.) Both requests and responses consist of 481 bytes of mixed data (text and numeric).

Chart displaying transactions per second on the vertical axis and percent CPU on the horizontal axis



The arrow in the graph shows the last level of CPU use at which transactions using **SelectionHint** and transactions not using **SelectionHint** ran at a similar number of TPS. Pushing the load beyond this level causes a drop in throughput. Up to the 85% CPU load level, you will see the same response time for both types of transactions. Using the **SelectionHint** property provides additional flexibility with only a very small cost in maximum throughput level (15%). On the other hand, when pushing 680 TPS through the server, not using **SelectionHint** drops the CPU from above 80% to a comfortable 60% level. These CPU cycles can be used to process the business logic on the middle tier.

See Also

Other Resources

Remote Environment Selection Using the SelectionHint Property

Performance Improvements in Host Integration Server

The Windows-initiated processing (WIP) feature of Host Integration Server, when installed on Windows Server 2003, shows the following performance improvements when compared to Host Integration Server 2000 SP1 installed on Windows 2000 Server:

- CICS over SNA performance is higher by up to 12%.
- CICS and IMS over TCP performance is higher by up to 30%.
- Performance for methods that contain **RecordSets** is higher by up to 77%.
- Enhanced Listener Message (ELM) performance is higher by up to 5% than Transaction Request Message (TRM) performance.
- Performance for persistent connection calls is higher than for the non-persistent calls by up to 56%.
- WIP .NET performance is generally about 17% of WIP COM; numbers for the .NET Framework methods that contain **DataTables** are within 30% of the COM methods that contain **RecordSets**
- The TCP ELM Link Persistent model delivers the best performance, with 1900 calls/sec for the light conversion methods.

The performance tests were conducted on a Fujitsu 4-processor server running Windows Server 2003, with all computers on a private 100-megabit network. The performance monitor was running on a separate computer.

See Also

Other Resources

Transaction Integrator Performance Guide

Technical Reference

The Technical Reference section contains areas that will aid your understanding of the Host Integration Server documentation terminology, and some specific help for issues you may encounter as you use Host Integration Server.

In This Section

- Glossary
- UI Help
- Administrators Reference

Glossary

3270

The information display system for IBM hosts (mainframes). The system includes terminals, printers, and controllers that enable a user to access host functions.

5250

The information display system for IBM AS/400 computers.

802.2

The logical link control protocol used for communication over a Token Ring or Ethernet network. The 802.2 protocol is an IEEE standard.

1-byte unsigned Integer

An integer data type that has a positive value ranging from 0 to 255.

2-byte signed Integer

An automation integer data type that can be either positive or negative. The most significant bit is the sign bit, which is 1 for negative values and 0 for positive values. The storage size of the integer is 2 bytes. A 2-byte signed integer can have a range from -32,768 to 32,767.

2PC

See **two-phase commit (2PC)**.

4-byte Real

Also referred to as a single-precision floating-point or Single. Single variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from –3.402823E38 to –1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. The type-declaration character for Single is an exclamation point (!).

4-byte signed Integer

An Automation integer data type that can be either positive or negative. The most significant bit is the sign bit, which is 1 for negative values and 0 for positive values. The storage size of the integer is 4 bytes. A 4-byte signed integer can have a range from -2,147,483,648 to 2,147,483,647.

8-byte Real

Also referred to as a double-precision floating-point or Double. Double data type variables are stored as 64-bit (8-byte) numbers. A Double variable is stored as a 64-bit (8-byte) number ranging in value from 1.79769313486232E308 to –4.94065645841247E-45 for negative values, from 4.94065645841247E-324 to 1.79769313486232E308 for positive values, and 0. The type-declaration character is a number sign (#).

3270 emulator

Software that enables a microcomputer to act as a 3270 terminal, displaying information from a host system (mainframe). Emulator software can also enable a desktop computer to send print jobs from a host system to a printer connected to the microcomputer.

3270 terminal emulation

The use of software that enables a microcomputer to act as a 3270 terminal, displaying information from a host system (mainframe). Emulation software can also enable a microcomputer to send print jobs from a host system to a printer connected to the microcomputer.

5250 emulator

Software that enables a microcomputer to act as a 5250 terminal interacting with an AS/400 system.

5250 terminal emulation

The use of software that enables a microcomputer to act as a 5250 terminal interacting with an AS/400 system.

A3270

The server transaction program for the APPC 3270 Terminal Emulator facility.

abend

Short for abnormal end. The premature ending of a program because of program error or system failure.

ACID

See **atomic, consistent, isolated, and durable (ACID)**.

acknowledgment required (ACKRQD)

A field in the header of a Status-Control message. If a Status-Control request has ACKRQD set in the message header, the recipient must supply a Status-Control response before the sender sends further messages or further Status-Control requests.

ACKRQD

See **acknowledgment required (ACKRQD)**.

ActiveX® Data Objects (ADO)

A data access interface that communicates with OLE DB-compliant data sources to connect to, retrieve, manipulate, and update data.

ACTLU

SNA command sent by the system services control point (SSCP) to a logical unit (LU) to activate a session and establish session parameters.

ACTPU

SNA command sent by the system services control point (SSCP) to activate a physical unit (PU), so that any logical units (LUs) controlled by this PU are available to the SNA network.

adapter

Refers to a circuit board, network card, and similar expansion devices with a specialized function, such as controlling a video display monitor or accessing a communications line. Not the same as a driver.

administration access

The level of access available to a user. The user may be granted or denied the right to use the interfaces (Host Integration Server Setup, SNA Manager, or the **snacfg** command) to read and change the configuration file, to start and stop services and connections, or to reset LUs.

ADO

See **ActiveX® Data Objects (ADO)**.

Advanced Peer-To-Peer Networking (APPN)

An extension to SNA that features (a) greater distributed network control that avoids critical hierarchical dependencies, thereby isolating the effects of single points of failure; (b) dynamic exchange of network topology information to foster ease of connection, reconfiguration, and adaptive route selection; (c) dynamic definition of network resources; and (d) automated resource registration and directory lookup. APPN extends the LU 6.2 peer orientation for end-user services to network control and supports multiple LU types, including LU 2, LU 3, and LU 6.2.

Advanced Program-to-Program Communications (APPC)

A means of enabling programs to communicate directly with each other, across a network or within a single system. APPC uses a type of logical unit called LU 6.2, and enables transaction programs (TPs) to engage in peer-to-peer communications in an SNA environment.

(1) The general term that characterizes the LU 6.2 architecture and its various implementations in products. (2) Refers to the LU 6.2 architecture and its product implementations as a whole or to an LU 6.2 product feature in particular, such as an APPC application programming interface. (3) A method for enabling programs to communicate directly with each other across a network or within a single system. APPC uses a type of LU called LU 6.2, and enables TPs to engage in peer-to-peer communications in an SNA environment.

AFTP

See **APPC File Transfer Protocol (AFTP)**.

AFTPD

The server transaction program for the APPC File Transfer Protocol facility.

aggregation

A composition technique for implementing component objects whereby a new object can be built using one or more existing objects that support some or all of the required interfaces of the new object.

alert

A message that indicates an abnormal event or a failure.

allocate

(1) The process that an operating system uses to respond to a request from a program to reserve memory for use by the program. (2) In Advanced Program-to-Program Communications (APPC), a verb that assigns a session to a conversation. *Contrast with***deallocate**.

American Standard Code for Information Interchange (ASCII)

A coding scheme that assigns numeric values to letters, numbers, punctuation marks, and certain other characters.

API

See **application programming interface (API)**.

APING

(1) The APPC Connectivity Tester facility. (2) The client transaction program for the APPC Connectivity Tester facility.

APINGD

The server transaction program for the APPC Connectivity Tester facility.

APPC

See **Advanced Program-to-Program Communications (APPC)**.

APPC File Transfer Protocol (AFTP)

(1) The client transaction program for the APPC File Transfer Protocol facility. (2) An interactive full-screen environment with a specific set of commands used to manage and transfer files between a client and server computer. (3) An API that provides APPC file transfer capabilities.

APPC mode

A collection of session properties used by LU 6.2-type logical units (LUs) as they carry on a session. A mode can be used by many LU pairs at the same time.

APPC mode name

The name used to represent a set of characteristics to be used in an APPC LU-LU session.

APPC verb

The mechanism by which a program accesses APPC. Each verb supplies parameters to APPC. *See also***Advanced Program-to-Program Communications (APPC)**.

application programming interface (API)

The set of programming language constructs or statements that can be coded in an application program to invoke the specific functions and services provided by an underlying operating system or service program.

application requester (AR)

(1) The source of a request to a remote relational database management system (DBMS). (2)The ODBC driver to DB2 connector that enables C and C++ applications to issue dynamic SQL queries and call DB2 stored procedures.

application TP

An application program that uses Advanced Program-to-Program Communications (APPC) to accomplish tasks for end-users and exchange data with other transaction programs (TPs) in an SNA environment.

APPN

See **Advanced Peer-To-Peer Networking (APPN)**.

See **Advanced Peer-To-Peer Networking (APPN)**.

AR

See **application requester (AR)**.

array

A set of sequentially indexed elements which have the same intrinsic data type. Each element of an array has a unique identifying index number. A changes made to one element of an array does not affect the other elements.

ASCII

See **American Standard Code for Information Interchange (ASCII)**.

assembly

A collection of functionality built, versioned, and deployed as a single implementation unit (one or multiple files). An assembly is the primary building block of a .NET Framework application. All managed types and resources are marked either as accessible only within their implementation unit or as exported for use by code outside that unit. In the common language runtime, the assembly establishes the name scope for resolving requests and the visibility boundaries are enforced. The runtime can determine and locate the assembly for any running object because every type is loaded in the context of an assembly.

assembly cache

A machine-wide code cache used for side-by-side storage of assemblies. There are two parts to the cache. First, the global assembly cache contains assemblies that are explicitly installed to be shared among many applications on the computer. Second, the download cache stores code downloaded from Internet or intranet sites, isolated to the application that triggered the download so that code downloaded on behalf of one application or page does not impact other applications. *See also***global assembly cache (GAC)**.

asynchronous verb completion

Processing of an SNA verb where the initial API call returns immediately, so that the normal operation of the program is not blocked while processing completes. When the verb completes, the application is notified through a Microsoft® Windows® message or event. *Contrast with***synchronous verb completion**.

atomic, consistent, isolated, and durable (ACID)

An acronym that describes the four key properties required of any Windows-based transaction:

- **Atomic.** Each transaction must execute completely or not at all.
- **Consistent.** The structural integrity of the transaction database must be maintained.
- **Isolated.** A transaction cannot access data that is already involved in a transaction.
- **Durable.** The TP data must be stored securely to enable recovery of the transaction results.

Note
A mainframe-based transaction program (TP) differs from a Windows-based transaction. A mainframe-based TP is a COBOL program that exists in the CICS or IMS environment and contains one or more mainframe transactions. A mainframe transaction might or might not meet the ACID properties.

atomicity

A feature of a transaction that indicates that either all actions of the transaction happen or none happens.

auditing

Tracking the activities of administrators and users by recording selected types of events (for example, the changing of the configuration file) in the security log of a computer running Host Integration Server.

authentication

The process of determining the identity of a user attempting to access a system. For example, passwords are commonly used to authenticate users.

automatic partnering

A setting for APPC LUs and modes that enables LU-LU pairs (with assigned modes) to be generated automatically by Host Integration Server. Each time a new APPC LU or mode is created with automatic partnering enabled, Host Integration Server searches for existing LUs and modes that also have automatic partnering enabled. Host Integration Server then uses all available automatic partners to create as many unique LU-LU pairs as possible, each pair containing a remote LU, local LU, and assigned mode. Disabling an automatic partner setting after an LU or mode has been created does not remove that LU or mode from LU-LU pairs already generated.

automatic transaction

A transaction that is created by the COM+ run-time environment for an object based on the transaction attribute of a component.

Automation

Automation is COM-based technology that enables dynamic binding to COM objects at run time.

Automation client

Also called an Automation controller. An application that manipulates the objects, methods, and properties of another application (the Automation server) through Automation.

Automation object

An object that is exposed to other applications or programming tools through Automation interfaces.

Automation server

An application that enables its objects, methods, and properties to be controlled by other applications through Automation.

-B-

backup configuration file

An extra copy of the configuration file, saved by using the **File** menu **BackupConfiguration** command in SNA Manager. The default extension for backup configuration file names is .SNA.

backup server

A computer running Host Integration Server, and designated as a backup server, on which the configuration file is replicated by Host Integration Server. Host Integration Server loads the copy of the configuration file located on a backup server if the primary server goes down. One or more computers running Host Integration Server can operate as backup servers.

Base

A part of each Host Integration Server component that provides the operating environment for the core functions of that component. The Base passes messages between components and provides functions common to all components, such as diagnostic tracing.

base client

A client that runs outside the COM+ run-time environment, but that instantiates COM+ objects.

base process

An application process in which a base client executes. A base client runs outside the COM+ run-time environment and instantiates COM+ application objects.

basic conversation

In APPC, a conversation type generally used by applications that provide services to other local applications. Basic conversations provide a greater degree of control over the transmission and handling of data than mapped conversations. *See also***mapped conversation**.

basic transmission unit (BTU)

A standard unit of information transmitted over an SNA network. A BTU consists of the transmission header (TH), the request/response header (RH), and the request/response unit (RU). The maximum size of the BTU is controlled in VTAM by the MAXDATA= parameter and in Host Integration Server by the Max BTU Length parameter.

batch job

A predefined sequence of programs that can be run through the Job Entry Subsystem or through an automated scheduling

system. Each program that runs as part of the sequence is considered a batch step. Typically data is passed from one step to the next through temporary or permanent files on the file system.

batch step

An application program that is executed as part of a larger batch job. Typically, data is read from and written to temporary or permanent files on the files systems.

BBI

See **begin bracket indicator (BBI)**.

BBIUI

See **begin basic information unit indicator (BBIUI)**.

BCI

See **begin chain indicator (BCI)**.

begin basic information unit indicator (BBIUI)

Bit 5 of Flag 2 of a Status-Control message. BBIUI is set on a Status-Control message corresponding to an outbound SNA request with BBIU (begin basic information unit). It is supplied solely for the use of SNA server components. Your application should not attempt to use it.

begin bracket indicator (BBI)

Bit 4 of Flag 1 of a Status-Control message. BBI is set if the chain carries BB (begin bracket). Note that this does not necessarily indicate that the bracket has been initiated.

begin chain indicator (BCI)

Bit 1 of Flag 1 of a Status-Control message. BCI is set if the message starts a chain.

blocking

A method of operation in which a program that issues a call does not regain control until the call completes. *See also***synchronous verb completion**.

Boolean expression

An expression that can be evaluated either true (nonzero) or false (0). You can use the keywords True and False to supply the values of -1 and 0, respectively. The field data type Yes/No is Boolean and has the value of -1 for Yes and 0 for No.

bounded

Refers to recordsets or arrays. The last input parameter or the last output parameter in a method can be bounded. This means its actual size can vary from zero to the maximum number of elements (in an array) or rows (in a recordset) specified at design time.

bracket

A chained set of RUs and their responses, which together make up a transaction between two LUs. One bracket must be finished before another can be started.

BTU

See **basic transmission unit (BTU)**.

business rule

The combination of validation edits, logon verifications, database lookups, policies, and algorithmic transformations that constitute an enterprise's way of doing business. Also known as business logic.

byte

A unit of information consisting of eight bits. A byte, or binary term, is the smallest collection of bits that can be accessed directly. The integer value of a byte can range from 0 to 255.

-C-

caller

A client that invokes a method of an object. The caller of an object is not necessarily the creator of an object. For example, client A could create object X and pass this reference to client B, and then client B could use that reference to call a method of

object X. In this case, client A is the creator, and client B is the caller. *See also***creator**.

catalog

In Windows 2000 and later, the catalog is the COM+ application data store that maintains configuration information for components, COM+ applications, and roles. You can administer the catalog by using TI Manager.

CDI

See **change direction indicator (CDI)**.

CEI

See **chain ending indicator (CEI)**.

chain

A series of related messages or data packets that are transmitted consecutively and are treated as a single entity forming a complete message.

change direction indicator (CDI)

Bit 6 of Flag 1 of a Status-Control message. CDI is set if chain carries change direction (CD).

Channel

A channel-attached connection to a host system.

characteristics

A set of internal values maintained by CPI-C for each conversation. They can affect the operation of the entire conversation or of specific calls.

CICS

See **Customer Information Control System (CICS)**.

class

A type that defines the interface of a particular type of object. A class defines the properties of the object and the methods used to control the behavior of an object.

class factory

An object that implements the **IClassFactory** interface, which enables it to create objects of a specific class.

class ID (CLSID)

A universally unique identifier (UUID) that identifies a COM component. Each COM component has its CLSID in the Windows registry so that it can be loaded by other applications.

client

A computer or a software component using services available through Host Integration Server. To run applications such as a 3270 emulator, the client uses the Host Integration Server computer to gain access host or peer systems on the SNA or TCP/IP network.

client/server

A distributed application model in which client applications request services from a server application. A server can have many clients at the same time, and a client can request data from multiple servers. An application can be both a client and a server.

CLSID

See **class ID (CLSID)**.

coaxial cable

A cable that consists of a conductor within another conductor, with insulation between the two conductors. The inner conductor is usually a small copper tube or wire, and the outer conductor is usually copper tubing or copper braid. It is the common medium used to connect LANs and 3270 devices. The maximum distance that a coaxial cable can be run between a 3270-type cluster controller and peripheral devices is 5,000 feet (1,500 meters).

code page

A table that associates specific ASCII or EBCDIC values with specific characters.

COM

See **Component Object Model (COM)**.

COM+

See **Component Services (COM+) component**; **Component Services (COM+) object**.

COMMAREA

An area of memory in the mainframe used for communications and accessible to various programs. It is similar to a data structure that contains both input parameters and return data.

Common Programming Interface for Communications (CPI-C)

A set of C-language routines that applications distributed across an SNA network can use to work together. Through CPI-C, distributed applications on computers communicating as peers can exchange data to accomplish a processing task, such as querying a remote database or copying a remote file.

An evolving application programming interface (API), embracing functions to meet the growing demands from different application environments and to achieve openness as an industry standard for communications programming. CPI-C provides access to interprogram services such as (a) sending and receiving data, (b) synchronizing processing between programs, and (c) notifying a partner of errors in the communication.

Common Service Verb (CSV)

An application programming interface (API) that provides ways of tracing, translating characters, and sending network management information to a host. Each verb supplies parameters to CSV.

communications controller

A device that directs the transmission of data over a network (for example, the IBM 3725 front-end processor).

COMP-1

Specified for internal floating-point items (single precision). The items are four bytes long. The sign is contained in the first bit of the leftmost byte, and the exponent is contained in the remaining seven bits of that byte. The last three bytes contain the mantissa.

COMP-2

Specified for internal floating-point items (double precision). The items are eight bytes long. The sign is contained in the first bit of the leftmost byte, and the remaining seven bits of that byte contain the exponent. The remaining seven bytes contain the mantissa.

COMP-3

Specified for internal decimal items. In storage, these items appear in packed decimal format. There are two digits for each character position (byte), except for the trailing character position (byte), which is occupied by the low-order digit and sign. The item can contain only the digits 0 through 9, plus a sign (in the last position), representing a value not exceeding 29 decimal digits (15 bytes).

component

A discrete unit of code built on ActiveX® controls that deliver a well-specified set of services through well-specified interfaces. Components provide the objects that clients request at run time.

Component Object Model (COM)

An open architecture for cross-platform development of client/server applications based on object-oriented technology. Clients have access to an object through the interfaces implemented on the object. COM is language-neutral, so any language that produces COM components can also produce COM applications.

Component Services (COM+) component

A Component Object Model (COM) component that executes in the COM+ run-time environment. A COM+ component is commonly known as a COM+ application. A COM+ component must be a dynamic-link library (.dll) file that implements a class factory to create objects and that describes all of the interfaces of the component in a type library to facilitate standard marshaling.

Component Services (COM+) object

A Component Object Model (COM) object that executes in the COM+ run-time environment.

concurrency

The appearance of simultaneous execution of processes or transactions by interleaving the execution of multiple pieces of work.

configuration file

A file containing setup and configuration information for Host Integration Server. It defines servers, connections, LUs, users, and other items. The configuration file that is loaded when SNA Manager starts is called COM.CFG.

connection

The data communication path between a workstation or server and other computers on the SNA network. Host Integration Server offers a variety of connection types:

- 802.2 (Token Ring or Ethernet)
- Synchronous Data Link Control (SDLC)
- X.25
- Distributed function terminal (DFT)
- Channel
- Twinax

connection object

In AFTP, a connection (not necessarily active) to a partner computer.

connectivity

(1) The capability of a system or device to be attached to other systems or devices without modification. (2) The capability to attach a variety of functional units without modifying them.

connectivity option

A type of connection hardware and software through which one computer communicates with other computers.

consistency

A state in which durable data matches the state expected by the business rules that modified the data.

constructor

In C, a special initialization function that is called automatically whenever an instance of a class is declared. This function prevents errors that result from the use of uninitialized objects. The constructor has the same name as the class itself and cannot return a value.

contention loser

In an APPC LU-LU session, the LU that cannot start a conversation with its partner LU (the contention winner) without first requesting permission of the partner LU. *See also***contention winner**.

contention winner

In an APPC LU-LU session, the LU that can start a conversation with its partner LU (the contention loser). If parallel sessions between the two LUs are being used, each LU may be the contention winner for some sessions and the contention loser for other sessions. *See also***contention loser**.

context

The state that is implicitly associated with a given COM+ object. The context contains information about the execution environment of an object, such as the identity of the creator of an object and, optionally, the transaction encompassing the work of the object. The context of an object is similar in concept to the process context that an operating system maintains for an executing program. The COM+ run-time environment manages a context for each object.

control point

A node or other SNA component that controls network resources and coordinates the activation of sessions.

controller

A device that directs the transmission of data over a network (for example, the IBM 3725 front-end processor).

conversation

The process used by network-based applications to communicate with each other and to exchange data to accomplish processing tasks. (1) A logical connection between two transaction programs using an LU 6.2 session. Conversations are delimited by brackets to gain exclusive use of a session. (2) The interaction between TPs carrying out a specific task. Each conversation requires an LU-LU session. A TP can be involved in several conversations simultaneously. *See also* **basic conversation**; **mapped conversation**.

conversation characteristics

Internal API values that define the overall operation for a conversation or for a specific call. *See also***application programming interface (API)**; **conversation**.

conversation ID

A unique identifier for a conversation between two transaction programs (TPs).

CPI-C

See **Common Programming Interface for Communications (CPI-C)**.

creator

A client that creates an object provided by a component (using **CreateObject**, **CoCreateInstance**, or the **CreateInstance** method). When a client creates an object, it is given an object reference that can be used to call the methods of that object. *See also***caller**.

CSV

See **Common Service Verb (CSV)**.

Currency

An 8-byte, fixed-point data type that is useful for calculations involving money or for fixed-point calculations in which accuracy is extremely important. This data type is used to store numbers with up to 15 digits to the left of the decimal point and 4 digits to the right. The type-declaration character in Microsoft® Visual Basic® is an at sign (@). Currency can range from –922,337,203,685,477.5808 to 922,337,203,685,477.5807.

current directory

The first directory in which the operating system looks for programs and data files and stores files for output.

Customer Information Control System (CICS)

An IBM transaction processing program that provides an environment on IBM mainframes in which applications can communicate with terminals or other applications.

-D-

DACTLU

An SNA command that is sent to deactivate the session between the system services control point (SSCP) and a logical unit (LU).

DACTPU

An SNA command that is sent to deactivate the session between the system services control point (SSCP) and a physical unit (PU).

data link control (DLC)

In SNA, the protocol stack layer that transmits messages across links and manages link-level flow and error recovery.

data set members

Members of partitioned data sets that are individually named elements of a larger file that can be retrieved by name.

data source name (DSN)

The name that applications use to request a connection to an ODBC data source. DSN also means Data Set Name on the mainframe.

database

(1) A collection of data with a given structure for accepting, storing, and providing on demand data for multiple users. (2) A collection of interrelated data organized according to a database schema to serve one or more applications. (3) A collection of data fundamental to a system. (4) A collection of data fundamental to an enterprise.

Date

An 8-byte, real data type used to store dates and times as a real number. Variables are stored as 64-bit numbers. The value to the left of the decimal represents a date, and the value to the right of the decimal represents a time. The Date data type can range from January 1, 1000 to December 31, 9999.

DCOM

See **distributed COM (DCOM)**.

DDM

See **distributed data management (DDM)**.

deallocate

(1) The process an operating system uses to free memory that has been previously allocated by a program. (2) In Advanced Program-to-Program Communications (APPC), a verb that ends a conversation. *Contrast with***allocate**.

Decimal

A data type that stores a signed, exact numeric value described as the number of digits appearing before and after the decimal point, with a maximum of 29 total digits. All possible digits cannot be represented if you are using the maximum number of digits.

default

The value that is automatically used if nothing is specified.

dependent local APPC LU

A local logical unit (LU) that enables Advanced Program-to-Program Communications (APPC) with a peer system, but only through a host (mainframe) system. The type of LU used in dependent APPC is LU 6.2.

DFT

See **distributed function terminal** (**DFT)**.

digit

In COBOL, any of the numerals from 0 through 9 not used in reference to any other symbol.

direct caller

The identity of the process (base client or server process) calling into the current server process.

direct creator

The identity of the process (base client or server process) that directly created the current object.

directory

(1) A list of files that are stored on a disk or diskette. A directory also contains information about the files such as size and date of last change. (2) A named grouping of files in a file system.

display emulation

A feature that enables a personal computer to emulate an IBM 3278 or 3279 terminal. *See also***emulation**.

display model

One of several different sizes of display:

- Model 2 is 24 lines by 80 characters
- Model 3 is 32 lines by 80 characters

- Model 4 is 43 lines by 80 characters
- Model 5 is 27 lines by 132 characters

display session

A 3270 emulation session between a networked personal computer and a host. The session is used to emulate a 3278 or 3279 display. Also called a host display session.

DISPLAY verb

An APPC verb that returns configuration information and current operating values for a computer running Host Integration Server.

distributed COM (DCOM)

An object protocol that enables COM components to communicate directly with each other across a network. Because DCOM is language-neutral, any language that uses COM components can also produce DCOM applications.

distributed data management (DDM)

A function of the operating system that enables an application program or user on one system to use database files stored on remote systems. A communications network must connect the systems, and the remote systems must also be using DDM.

distributed function terminal(**DFT)**

A type of intelligent terminal supported by IBM 3270 control units, in which some of the terminal's functions are controlled by the terminal and some by the control unit. Enables multiple sessions, and connects to host systems or to peer systems through host systems. DFT terminals are often connected using coaxial cable.

distributed processing

A form of information processing in which the work is performed by separate computers that are linked through a local or wide area network, using data-transfer mechanisms that enable different programs to use and share data.

distributed program call (DPC)

An AS/400 remote communication model.

Distributed Query Processor (DQP)

Enables queries to access multiple data sources on multiple servers, even SQL and DB2, and combine views, create data warehouses, and so on. DQP supports an extended version of the SQL language that permits users to qualify table names with the databases in which they exist. This gives users the capability to formulate queries that span multiple distributed databases.

Distributed Relational Data Architecture (DRDA)

A connection protocol for distributed relational database processing that is used by IBM relational database products. The DRDA protocol comprises protocols for communication between an application and a remote database, and communication between databases. The DRDA protocol provides the connections for remote and distributed processing. The DRDA protocol is built on the Distributed Data Management Architecture.

Distributed Transaction Coordinator (DTC)

A transaction manager that coordinates transactions spanning multiple resource managers. Work can be committed as an atomic transaction even if it spans multiple resource managers, even on separate computers.

distributed unit of work (DUW)

In DB2 UDB for AS/400, this is a method of accessing distributed relational data in which a user or application can, within a single unit of work, read and update data on multiple database management systems (DBMSs). The user or application directs each SQL statement to a particular DBMS for execution at the DBMS. Each SQL statement may access only one DBMS.

DL-BASE

The type of Base used by Host Integration Server 3270 emulation programs. It supports a single Host Integration Server component or a single user application and has entry points for initialization, sending messages, receiving messages, and termination. *See also***Base**.

DLC

See **data link control (DLC)**.

DLL

See **dynamic-link library (DLL)**.

DMOD

See **Dynamic Access Module (DMOD)**.

document type definition (DTD)

Can accompany a document, essentially defining the rules of the document, such as which elements are present and the structural relationship between the elements. It defines what tags can go in your document, what tags can contain other tags, the number and sequence of the tags, the attributes your tags can have, and optionally, the values those attributes can have.

DTDs help to validate the data when the receiving application does not have a built-in description of the incoming data. The DTD is declared within the document type declaration production of the XML file. With XML, however, DTDs are optional.

downstream connection

A connection that enables a computer running Host Integration Server to support communication between hosts and clients. Even though such clients do not use the Host Integration Server client/server interface, with a downstream connection they can access host connections available through a computer running Host Integration Server.

Host Integration Server offers several types of downstream connection:

- 802.2 (Token Ring or Ethernet)
- SDLC
- X.25

downstream LU

A logical unit (LU) used by clients to access a host connection through a computer running Host Integration Server. Such clients do not use the Host Integration Server client/server interface, but by using a downstream LU, can receive access to connections on a computer running Host Integration Server. A downstream LU uses a downstream connection, and passes information between the client and the host.

downstream system

A client such as an IBM Communications Manager/2 system that can access host connections available on a computer running Host Integration Server. Even though such clients do not use the Host Integration Server client/server interface, they can use a downstream connection and a downstream LU to communicate with the host through Host Integration Server. Host Integration Server passes the information between the downstream system and the host. With Host Integration Server, downstream systems appear to the host as logical units, not physical units.

DPC

See **distributed program call (DPC)**.

DPL-enabled

Compatible with the IBM Distributed Program Link (DPL) protocol.

DQP

See **Distributed Query Processor (DQP)**.

DRDA

See **Distributed Relational Data Architecture (DRDA)**.

DSN

See **data source name (DSN)**.

DTC

See **Distributed Transaction Coordinator (DTC)**.

DTD

See **document type definition (DTD)**.

duplex

Capable of simultaneously transmitting and receiving data. Also called **full-duplex** or **4-wire**. *Contrast with***half-duplex**.

durability

A state that survives failures.

DUW

See **distributed unit of work (DUW)**.

Dynamic Access Module (DMOD)

An SNA component that provides the communications facilities needed to pass messages between the Bases.

dynamic-link library (DLL)

A binary file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. The operating system maps a DLL to the address space of the calling process when the process starts or while it is running. It uses the .dll file extension.

-E-

EBCDIC

See **Extended Binary Coded Decimal Interchange Code (EBCDIC)**.

EBI

See **end bracket indicator (EBI)**.

EBIUI

See **end basic information unit indicator (EBIUI)**.

ECI

See **end chain indicator (ECI)**.

ELM

See **enhanced listener message (ELM)**.

emulation

A process whereby one device imitates another; for example, a personal computer can emulate a 3278 terminal. *See also***display emulation**.

end bracket indicator (EBI)

Bit 5 of Flag 1 of a Status-Control message. Set if chain carries end bracket (EB). Note that this does not indicate that the bracket has terminated.

end chain indicator (ECI)

Bit 2 of Flag 1 of a Status-Control message. Set if this message ends a chain.

enhanced listener message (ELM)

A streamlined, application-level protocol exchange sequence that sends to and receives from the host application a single data stream composed of a header followed by the application data.

ERI

See **exception response indicator (ERI)**.

Ethernet

An IEEE 802.3 standard for contention networks. Ethernet uses a bus or star topology and relies on the form of access known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD) to regulate communication line traffic. Network nodes

are linked by coaxial cable, by fiber-optic cable, or by twisted-pair wiring. Data is transmitted in variable-length frames containing delivery and control information and up to 1,500 bytes of data. The Ethernet standard provides for baseband transmission at 10 megabits per second.

event log

Host Integration Server records events involving communications hardware (for example, communications adapters) or software in the Windows Event Log. Events can include attempts to establish communication, successful establishment of sessions, failures of system components, attempts to use files that are damaged or missing, configuration problems, and responses from remote systems.

exception

An abnormal condition or error that occurs during the execution of a program and that requires the execution of software outside the normal flow of control.

exception request (EXR)

A request in which an intermediate component has detected an error and modified the request so the final destination is also aware of the error.

exception response indicator (ERI)

A specified response for a request. The response should be issued only if the request cannot be processed or if an error was encountered during processing.

exchange identification (XID)

An identifier that is exchanged between nodes on an SNA network, and that enables the nodes to recognize each other and to establish link and node characteristics for communicating. With Host Integration Server, there are two possible kinds of XIDs that can be exchanged: Format 0 XIDs (containing only basic information such as Node ID) and Format 3 XIDs (containing more detailed information such as Network Name and Control Point Name). *See also***Format 0 XID**; **Format 3 XID**.

EXR

See **exception request (EXR)**.

Extended Binary Coded Decimal Interchange Code (EBCDIC)

A coding scheme developed by IBM for use with its mainframe and AS400 computers as a standard method of assigning binary (numeric) values to alphabetic, numeric, punctuation, and transmission-control characters.

Extensible Markup Language (XML)

A specification developed by the World Wide Web Consortium (W3C) that enables designers to create customized tags beyond the capabilities of standard Hypertext Markup Language (HTML). While HTML uses only predefined tags to describe elements within the page, XML enables tags to be defined by the developer of the page. Tags for virtually any data item, such as a product or an amount due, can be used for specific applications. This enables Web pages to function as database records.

Extensible Stylesheet Language (XSL)

A style sheet format for Extensible Markup Language (XML) documents. XSL is used to define the display of XML in the same way that cascading style sheets (CSS) are used to define the display of Hypertext Markup Language (HTML).

-F-

fault isolation

Containing the effects of a fault within a component, rather than propagating the fault to other components in the system.

fault tolerance

The ability of a system to recover from an error, a failure, or a change in environmental conditions (such as loss of power). True fault tolerance provides for fully automatic recovery without disruption of user tasks or files, in contrast to manual means of recovery such as restoring data loss with backup files.

file transfer

The process of sending and receiving data files to and from computers.

fill type

A value that indicates whether programs will receive data in the form of logical records or as a specified length of data.

flow

A verb flows from one LU to another.

FMHI

See **function management header indicator (FMHI)**.

FMI

See **function management interface (FMI)**.

Format 0 XID

A type of XID that supplies minimal information about the node. Format 0 XIDs have a fixed length. They can be used for 3270 and LUA communication, and cannot be used for Advanced Program-to-Program Communications (APPC). *See also***exchange identification (XID)**, **Format 3 XID**.

Format 3 XID

A type of XID that supplies more detailed information about the node than a Format 0 XID. Format 3 XIDs have a variable length. They can be used for 3270 and LUA communication, and are the only type of XID that can be used for Advanced Program-to-Program Communications (APPC). *See also***exchange identification (XID)**; **Format 0 XID**.

full-duplex

Capable of simultaneously transmitting and receiving data. Also called **duplex** or **4-wire**. *Contrast with***half-duplex**.

full-duplex transmission

Two-way electronic communication that takes place in both directions simultaneously. Also called **duplex transmission** or **4-wire transmission**. *Contrast with***half-duplex transmission**.

fully qualified LU name

The two-part network address (network.lu) that uniquely identifies a destination (typically a user) in the network.

function management header indicator (FMHI)

Headers inserted into requests containing end-user data to convey control information.

function management interface (FMI)

An interface that provides applications with direct access to SNA data flow and information about SNA control flows by means of status messages. It is particularly suited to the requirements of 3270 emulation applications.

-G-

GAC

See **global assembly cache (GAC)**.

global assembly cache (GAC)

A machine-wide code cache that stores assemblies specifically installed to be shared by many applications on the computer. Applications deployed in the global assembly cache must have a strong name.

group

A set of one or more Windows 2000 or later user accounts.

-H-

half-duplex

Capable of only one direction of communication at a time, either receiving data or transmitting data, but not doing both at the same time. Also called **2-wire**. *Contrast with***full-duplex**.

half-duplex transmission

Two-way electronic communication that takes place in only one direction at a time. Also called **2-wire transmission**. *Contrast with***full-duplex transmission**.

HCD

See **host column description (HCD)**.

HE

See **host environment (HE)**.

high-level language application programming interface (HLLAPI)

An API that enables you to develop and run programmer-operator applications on IBM personal computers (or compatibles) that communicate with IBM mainframes using 3270 emulation.

HIP

See **host-initiated processing (HIP)**.

HLLAPI

See **high-level language application programming interface (HLLAPI)**.

host column description (HCD)

Maps AS/400 flat file data types to OLE DB data types. The HCD is an external file stored on the computer that enables administrators to describe the host record format. At run time, the OLE DB Provider for AS/400 and VSAM transparently converts the host data to computer data using the local HCD information.

host environment (HE)

An object that defines the network and hardware characteristics of the non-Windows software platform that initiates requests to the Windows platform. The host environment consists of the host environment name, host identification, network transport type, data conversion information, default method resolution criteria, and security credential mapping.

Host Integration Server

A Microsoft® software program that enables a personal computer to communicate with remote computers such as IBM mainframes, AS/400s, or other personal computers on a TCP/IP or SNA network.

host response time

The amount of time that a host computer takes to reply to a message sent to it by a client computer. Host response time is measured from the moment that the personal computer sends the message until one of the following events: the client computer receives data back from the host, the host unlocks the client computer's keyboard, or the host enables the client computer to send more data.

host system

A computer system (usually a mainframe) that controls interactions between it and the computers connected to it. A host system makes operating systems and applications available by way of Host Integration Server to computers running software for terminal emulation or for APPC.

In SNA terminology, a host is capable of sending an ACTPU command to Host Integration Server and sets up a PU-SSCP session with Host Integration Server.

host-addressable printer

A printer that is defined as a device associated with a logical unit (LU) configured as LU type 1 or 3 and that can support host printing as well as local printing.

host-initiated processing (HIP)

A non-Microsoft software platform (usually a mainframe or mid-range computer such as the AS/400) that can access and integrate its programs with the programs on a Windows server platform.

hot backup

(1) The ability to take systems online and offline without disrupting service. (2) A configuration in which one resource (such as a server running Host Integration Server software) can automatically handle sessions if another cannot. Such servers can provide hot backup for 3270, LUA, or downstream sessions through pools containing LUs from multiple servers. Servers running Host Integration Server software can provide hot backup for 5250 terminal emulation through the use of LU names that are the same on multiple servers.

-I-

I-frame

See **Information frame (I-frame)**.

identity

A COM+ application property page that specifies the user accounts authorized to use that application. You can set it to **Interactive user** (to authorize the current logged on user), to a specific user account, or to a group of users within a Windows domain.

IEEE

See **Institute of Electrical and Electronics Engineers (IEEE)**.

implicit incoming mode

A mode that defines the properties to use when Host Integration Server receives a request to start a session, and the mode named in the request is not recognized by Host Integration Server. An implicit incoming mode enables greater flexibility in starting sessions with remote systems.

For a session to be established, the incoming local LU name must be recognized by Host Integration Server. Then, the incoming remote LU must either be recognized explicitly or handled implicitly (if an implicit incoming remote LU has been configured). If the remote LU is recognized explicitly, but the mode is not recognized (as part of an LU-LU pair), Host Integration Server internally creates a new mode definition with the correct name, using the properties of the implicit incoming mode. Alternatively, if the remote LU is handled implicitly, Host Integration Server also handles the mode implicitly, by internally creating a mode, as described.

Note that an implicit incoming mode must be configured for any remote LU that will be used as an implicit incoming remote LU. An implicit incoming mode can be (but does not have to be) configured for remote LUs that will only be used explicitly.

implicit incoming remote LU

A remote APPC LU that defines the properties to use when Host Integration Server receives a request to start a session with a local LU, and the remote LU named in the request is not recognized by Host Integration Server. An implicit incoming remote LU that enables greater flexibility in starting sessions with remote systems.

Note that for a session to be established, the local LU name must be recognized by Host Integration Server. If the local LU name is recognized, but the remote LU name is not recognized as a partner for the local LU, Host Integration Server internally creates a new remote LU definition with the correct name, using the properties of the implicit incoming remote LU.

In SNA Manager, before a remote APPC LU can be used as an implicit incoming remote LU, an implicit incoming mode must be configured for it.

IMS

See **Information Management Systems (IMS)**.

independent local APPC LU

A local logical unit (LU) that enables Advanced Program-to-Program Communications (APPC) with a peer system without involving a host (mainframe) system. The type of LU used in independent APPC is LU 6.2. An independent LU does not require a host system, but can work through one.

IND$FILE

IBM file transfer program that enables files to be transferred from a personal computer to the host and from the host to the personal computer. It operates in three host environments: CICS, VM/CMS, and MVS/TSO.

Information frame (I-frame)

A standard unit of information transmitted over an SNA network. For 802.2 or SDLC communication, an I-frame is equivalent to a BTU. *See also***basic transmission unit (BTU)**.

Information Management Systems (IMS)

A transaction processing monitor created and sold by IBM Corporation.

in-process component

A component that runs in a client's process space. This is typically a dynamic-link library (DLL).

instance

An object of a particular component class. Each instance has its own private data elements or member variables. A component instance is synonymous with object.

Institute of Electrical and Electronics Engineers (IEEE)

An organization that maintains the standards for the 802.x protocols used in communications on local area networks.

Integer

A fundamental Automation data type that holds integer numbers. An integer variable is stored as a 16-bit (2-byte) number ranging in value from –32,768 to 32,767. The type-declaration character is a percent sign (%) (ANSI character 37). In Microsoft® Visual Basic®, you can use integers to store Boolean (**True**/**False**) values.

interface

A group of logically related operations or methods that provides access to a component object.

Internet Packet Exchange/Sequenced Packet Exchange (IPX/SPX)

A set of protocols used by Novell NetWare networking software for communication across a network.

Internet Protocol (IP) routed network

A TCP/IP wide area network in which IP packets are propagated across the network through devices called IP routers.

invokable

Indicates the capability of a program to be started by another program. For example, an invokable APPC transaction program (TP) can be started in response to a request from another TP (the invoking TP).

invoked program

A program that has been activated by a call or verb. *See also***invoking program**.

invoked TP

A host transaction program (TP) started by:

- Another (the invoking) TP.
- A Transaction Integrator Automation server working in conjunction with the TI run-time environment and Microsoft Distributed Transaction Server (DTS) included in COM+.

invoking program

A program that uses a call or verb to activate another program. Also known as the calling program or the client. *See also***invoked program**.

invoking TP

A TP that initiates a conversation with another TP. The invoking TP starts the other TP by instructing the remote node to load the invokable TP.

IP routed network

See **Internet Protocol (IP) routed network**.

IPX/SPX

See **Internet Packet Exchange/Sequenced Packet Exchange (IPX/SPX)**.

isolation

A characteristic whereby two transactions running in parallel produce the illusion that there is no concurrency. It appears that the system is running one transaction at a time.

-J-

No terms.

-K-

No terms.

-L-

LAN

See **local area network (LAN)**.

LE

See **local environment (LE)**.

leased SDLC line

A dedicated telecommunications line using SDLC. *See also***Synchronous Data Link Control (SDLC)**.

link service

The software component of Host Integration Server that communicates with the device driver for a particular communication adapter (802.2, SDLC, X.25, DFT, Channel, or Twinax).

listener

A local environment associated with an application, where the local environment monitors the TCP/IP or SNA network for requests to the application.

load balancing

Distribution of the processing load among several servers carrying out network tasks to increase overall network performance.

local account

An account provided in a local domain for a user whose regular account is not in a trusted domain. Local accounts cannot be used to log on interactively. Local accounts created in one domain cannot be used in trusted domains.

local area network (LAN)

A high-speed communication system consisting of hardware (computers and peripherals) and software (programs and data files) that are interconnected by cable in a way that enables these resources to be shared. The connected devices are located within a limited geographic area such as a building or campus.

local environment (LE)

An object that defines the endpoint on a Windows computer that accepts incoming requests from a non-Windows software platform. The local environment consists of the local environment name, network transport type, network transport class, and endpoint identification.

local LU

In an APPC or CPI-C conversation, the logical unit (LU) on the local end. *Contrast with***partner LU** and **remote LU**.

local LU alias

The name by which a local logical unit (LU) is known to the local transaction program (TP).

local node

The software component of Host Integration Server that interacts with clients and other nodes on the SNA network.

local printer

A printer that is attached directly to a personal computer.

local program

In CPI-C, the program on the local end of the conversation. *Contrast with* partner program.

local TP

In an Advanced Program-to-Program Communications (APPC) or Common Programming Interface for Communications (CPI-C) conversation, the transaction program (TP) on the local end. *Contrast with***partner TPs** and **remote transaction program**. *See also***local LU**.

locality

A base and the components within it; that is, a Host Integration Server executable program.

locality, partner, index (LPI)

An LPI address that is used to identify each end of a connection. It has three components: locality (L), partner (P), and index (I).

logical unit (LU)

(1) A type of network-accessible unit that enables users to gain access to network resources and communicate with each other. (2) A preset unit containing all of the configuration information needed for a user, program, or downstream system to establish a session with a host or peer computer. *See also***LU alias**; **LU name**; **LU pool**.

logical unit application (LUA)

A conventional LU application, or the interface that these applications use. LUA enables workstations to communicate with host applications using LU 0, 1, 2, or 3 protocols.

LPI

See **locality, partner, index (LPI)**.

LPI address

Used to identify each end of a connection between two partners. It can have three components: L identifies the locality, P identifies the partner within the locality, and I identifies a logical entity within the partner. *See also***locality**; **partner**.

LU

See **logical unit (LU)**.

LU alias

A string that identifies an APPC or CPI-C logical unit (LU) to transaction programs (TPs) in the same organizational unit (OU). An LU alias is used only locally by Host Integration Server, but it also can be used by any program in the OU of the host mainframe system. *See also***LU name**.

LU name

For 3270 or LUA communication, a name that identifies a logical unit (LU). For independent APPC or CPI-C, a name that (when used with the network name) identifies an LU to other components on an SNA network. For dependent APPC or CPI-C, a name that identifies an LU to local software, such as the Windows Event Viewer. *See also***LU alias**.

LU pool

A number of logical units (LUs) of the same type that are made available as a group. A user or LU application addressing the pool will connect to the next available LU in the pool for that session only. *See also***logical unit (LU)**.

LU type

Logical unit type. A subset of the SNA protocol that characterizes the communication between two LUs.

LU type 0

A logical-unit protocol with minimal constraints, on which special applications can be built for SNA.

LU type 1

A logical-unit protocol used by a host application communicating with a printer, sending data that conforms to the 3270 SNA Character String (SCS) definition.

LU type 2

A logical-unit protocol used by a host application communicating with a 3270-type display terminal, using the SNA 3270 data stream.

LU type 3

A logical-unit protocol used by a host application communicating with a printer, sending data that is 3270 data stream compatible (DSC).

LU type 6.2

A logical-unit protocol used by two applications or transaction programs (TPs) communicating as peers in an SNA environment. LU 6.2 works in combination with node type 2.1 to provide Advanced Program-to-Program Communications (APPC) using independent LUs. LU 6.2 also works with node type 2.0 to provide APPC with dependent LUs.

LU-LU session

A logical, two-way exchange between two logical units (LUs) over a specific connection for a specific amount of time.

LUA

See **logical unit application (LUA)**.

-M-

MAC address

A 12-byte hexadecimal address used by the media access control (MAC) layer of an 802.2 connection. It corresponds to the VTAM MACADDR= parameter and to the Remote Network Address parameter for an 802.2 connection with Host Integration Server.

management object

A TI component that manages or provides access to administration information. Typically, management objects are visible only when errors or messages are reported or placed in the Windows Event Log.

mapped conversation

A conversation in which the sending program sends one logical record at a time and the receiving program receives one record at a time. *See also* **conversation**.

marshaling

The process of packaging and sending interface method parameters across thread or process boundaries.

member server

A server that does not contain a configuration file. One or more servers can operate as member servers. The other types of servers are the primary server and backup servers.

method

A procedure (function) that acts on an object.

Microsoft .NET

Microsoft® .NET is a set of software technologies for connecting information, people, systems, and devices. This new generation of technology is based on Web services—small building-block applications that can connect to each other as well as to other, larger applications over the Internet.

mode

A collection of session properties used by LU 6.2-type logical units (LUs) as they carry on a session. A mode can be used by many LU pairs at the same time.

mode name

The name used by the initiator of a session to designate the characteristics desired for the session, such as traffic pacing values, message-length limits, Sync Point and cryptography options, and the class of service within the transport network.

model

One of several different sizes of display:

- Model 2 is 24 lines by 80 characters
- Model 3 is 32 lines by 80 characters
- Model 4 is 43 lines by 80 characters
- Model 5 is 27 lines by 132 characters

Messaging-oriented middleware

Messaging-oriented middleware (MOM) is a set of products that connects applications running on different systems by sending and receiving application data as messages. Examples are RPC, CPI-C, and message queuing.

multidrop

A connection in which one primary node communicates with multiple secondary nodes concurrently over the same physical transmission medium.

multiple sessions

In CPI-C, two or more concurrent sessions with different partner LUs. *Seealso***LU-LU session**.

Multiple Virtual Storage (MVS)

An operating system for large IBM mainframe computers. Implies MVS/370, the MVS/XA product, and the MVS/ESA product.

MVS

See **Multiple Virtual Storage (MVS)**.

-N-

NAU

See **network addressable unit (NAU)**.

NC

See **network control (NC)**.

NCP

See **Network Control Program (NCP)**.

.NET Framework

An integral Microsoft® Windows® component for building and running the next generation of applications and XML Web services.

NetView

A reporting system that runs on an IBM host (mainframe), forwarding alerts and other information back and forth between the host and personal computers, and other network addressable units that connect to the host.

NetView alert

A message sent to the NetView reporting system, indicating an abnormal event or a failure.

NetView user alert

A message sent by a 3270 user to a host system operator through NetView, requesting an action such as mounting a tape or changing forms on a printer. Also called user alert.

NetWare

A collection of networking software products from Novell, Inc.

network

Computer systems, controllers, terminals, and software connected in a way that enables them to communicate with each other.

network addressable unit (NAU)

The basic functional entities in an SNA environment that are the source or destination of all information flowing within the SNA network. The NAU can be a logical unit (LU), a physical unit (PU), or a system services control point (SSCP).

network control (NC)

A set of SNA-defined requests and responses used to control explicit and virtual routing.

Network Control Program (NCP)

An IBM program that supports communication controllers in single-domain, multiple-domain, and interconnected networks.

Network Management Vector Transport (NMVT)

SNA message containing network or system management information.

network name

A name identifying an SNA network. The network name is used in combination with other identifiers, either a control point name (to identify a control point or node) or an LU name (to identify an APPC LU, particularly an independent local APPC LU). The combination of a network name with a control point name is sometimes called a network qualified control point name. The combination of a network name with an LU name is sometimes called a fully qualified network name.

NMVT

See **Network Management Vector Transport (NMVT)**.

node

(1) A server, controller, workstation, printer, or other processor that implements SNA functions. SNA defines three kinds of nodes: the host subarea node, which functions to control and manage a network; the communication controller subarea node, which routes and controls data flow through the network; and peripheral nodes, which include printers, workstations, cluster controllers, and distributed processors.

(2) A branch on a navigation tree.

node type 2.1

An SNA component, such as an intelligent terminal or a personal computer, that works together with LU type 6.2 to support peer-to-peer communications, allowing the logical units (LUs) to function independently from the host.

null

A value that indicates missing or unknown data.

-O-

object

A run-time instance of a Component Object Model (COM) component. An object is created by a component's class factory. Object is synonymous with instance.

object variable

A variable that contains a reference to an object.

OCCURS DEPENDING ON

Code syntax that specifies variable-length tables. This is the COBOL version of an array that contains a variable number of elements.

OCCURS *fixed times*

Code syntax that specifies fixed-length tables. This is the COBOL version of an array.

ODBC

See **open database connectivity (ODBC)**.

ODBC resource dispenser

A resource dispenser that manages pools of database connections for COM+ components that use the standard open database connectivity (ODBC) programming interfaces.

open database connectivity (ODBC)

A set of standards that enables universal access to relational data, including Microsoft relational databases and mainframe databases.

open transaction management architecture (OTMA)

A high-performance, connectionless protocol used by IMS to communicate efficiently with Multiple Virtual Storage (MVS) applications without using the SNA protocol.

operator-loaded TP

An invokable transaction program (TP) that is manually loaded and started by an operator.

original caller

The identity of the base client that initiates an activity.

original creator

The identity of the base client that created the current object. The original caller and original creator are different only if the original creator passed the object to a different base client. *See also***original caller**.

OS/390

The IBM operating system for the IBM S/390 family of enterprise servers and that includes and integrates functions previously provided by other IBM software products such as the MVS operating system.

OS/400

The IBM operating system for the IBM AS/400.

OTMA

See **open transaction management architecture (OTMA)**.

out-of-process component

A component that runs in a separate process space from its client.

-P-

pacing receive count

The maximum number of frames for the local logical unit (LU) to receive from the partner LU before the local LU sends a response.

pacing send count

The maximum number of frames for the local logical unit (LU) to send without receiving an SNA pacing response from the partner LU.

packet

A transmission unit of fixed maximum size, used as the basic unit on a packet-switching network. A packet contains both a header and data.

In data communication, a sequence of binary digits, including data and control signals, that is transmitted and switched as a composite whole. The data, control signals, and possibly error control information are arranged in a specific format.

packet switching

A message-delivery technique in which small units of information (packets) are relayed through stations in a computer network along the best route currently available between the source and the destination. Packet-switching networks are considered to be fast and efficient. The protocol used on packet-switching networks is X.25. *See also* **X.25**.

parallel sessions

Multiple concurrent sessions between a pair of LU 6.2-type logical units (LUs), allowing multiple operations to be performed simultaneously.

parameter

A variable used as input to a program, operating system, or API to govern how systems, programs, or functions perform.

partitioned data set (PDS)

A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partner

An addressable component of a locality; that is, code to which messages can be sent. *See also***locality**.

partner LU

In an APPC or CPI-C conversation, the LU on the far end. The partner LU serves the partner Transaction Processor. *Contrast with***local LU**; *see also***remote LU**.

partner LU alias

A name that identifies a partner logical unit (LU) to partner transaction programs (TPs).

partner LU name

A name that identifies a partner logical unit (LU) to other LUs on the LU 6.2 session.

partner program

For CPI-C, the program receiving the CPI-C call.

partner TPs

Two transaction programs (TPs), residing on the same or separate nodes that are configured to communicate with each other. Partner TPs use partner LUs.

password

A string of characters that a user, a program, or a computer operator must specify to meet security requirements before gaining access to a system and to the information stored within it.

path

(1) In SNA, the series of nodes and communications links over which data must travel from one LU to another. (2) A sequence of folders that identify the location of a file. (3) A path exists between two localities when the DMODs in the localities can successfully pass messages between them. A path must exist between two localities before a connection can exist between partners in these localities. *See also***Dynamic Access Module (DMOD)**; **locality**.

pattern-matching character

A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace a pattern-matching character. *Synonymous with* wildcard character.

PC Support

A set of IBM programs that helps personal computer users access, share, and store information on an AS/400.

PDS

See **partitioned data set (PDS)**.

peer system

A mainframe, midrange, or personal computer that communicates with another computer as an equal partner, with both computers sharing control over the communication.

peer-to-peer

A type of communication in which two systems communicate as equal partners sharing the processing and control of the exchange, as opposed to host-terminal communication in which the host does most of the processing and controls the exchange.

permanent virtual circuit (PVC)

A type of circuit used by an X.25 connection, in which the circuit is constantly active, and the destination address is preset.

permissions

Settings that grant or deny a particular kind of access to a particular file, folder, or other object. For example, granting read permission but denying write permission for File1.ext for Domain Admins means that the members of Admins group can read but not change File1.ext.

physical unit (PU)

A network-addressable unit that provides the services needed to use and manage a particular device, such as a communications link device. A PU is implemented with a combination of hardware, software, and microcode.

PIC S9(4) COMP Integer

A 16-bit COBOL data type that represents signed arithmetic operations occupying 2 bytes of storage. This is normally analogous to an Integer data type in Microsoft® Visual Basic® and a Short Integer in C when referring to 32-bit. It can take on values from –9999 to +9999 or –32768 to +32767. It is similar to a Short in C.

PIC S9(9) COMP Integer

A 32-bit COBOL assignment statement to represent signed arithmetic operations that occupy 4 bytes of storage. It can take on values from –999999999 to +999999999 or –2147483648 to +2147483647 depending on compiler options. It is similar to a Long Integer in C.

PIC X

Specifies a single COBOL EBCDIC character.

PIC X No Translation

A character string handled like binary data. There is no translation from EBCDIC to Unicode or from Unicode to EBCDIC.

PICTURE clause

Specifies the general characteristics and editing requirements of an elementary item. The PICTURE character string is made up of COBOL characters used as symbols and can contain a maximum of 30 characters.

pipe

A portion of memory that can be used by one process to pass information along to another.

PLU

See **primary logical unit (PLU)**.

pool

See LU pool.

pooling

A performance optimization based on using collections of pre-allocated resources, such as objects or database connections. Pooling results in more efficient resource allocation.

primary logical unit (PLU)

On an SNA session, the LU on the node that sent the session activation request.

primary server

The server designated to contain the primary configuration file. There can be only one primary server active in a subdomain. *See also***backup server**.

printer emulation

The ability of a personal computer-type printer to emulate a 3287 or 4224 printer to print host data.

printer session

A 3270 emulation session between a host and a local area network printer connected to a personal computer. The printer emulates the type of printer normally used by a host system.

private assembly

An assembly that is available only to clients in the same directory structure as the assembly. *See also***assembly**.

ProgID

See **programmatic identifier (ProgID)**.

programmatic identifier (ProgID)

A name that identifies a COM component. For example, a ProgID could be Bank.MoveMoney.

programmatic security

Procedural logic provided by a component to determine if a client is authorized to perform the requested operation. *See also***declarative security**.

protocol

(1) A set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. (2) In Open Systems Interconnection architecture, a set of semantic and syntactic rules that determine the behavior of entities in the same layer in performing communication functions. (3) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

proxy

An interface-specific object that provides the parameter marshaling and communication required for a client to call an application object that is running in a different execution environment, such as on a different thread or in another process. The proxy is located with the client and communicates with a corresponding stub that is located with the application object that is being called. In the case of TI, the TI run-time environment serves as the proxy to the mainframe transaction program (TP).

PU

See **physical unit (PU)**.

PU 2.0

In an SNA network, the component that defines controller and terminal-type resources similar to an IBM 3274 Control Unit.

PU 2.1

In an SNA network, a component such as an intelligent terminal or a personal computer that works together with logical unit (LU) type 6.2 to support peer-to-peer communications, allowing the LUs to function independently from the host.

PVC

See **permanent virtual circuit (PVC)**.

-Q-

QLLC

See **qualified logical link control (QLLC)**.

qualified logical link control (QLLC)

A protocol that permits SNA sessions to occur over X.25 networks.

queued TP

An invokable transaction program (TP) that can be started by only one incoming allocate command at a time. Incoming allocate commands that arrive while the queued TP is running do not start the program again, but are queued until the program issues another **RECEIVE_ALLOCATE** or until it finishes execution.

-R-

race condition

A condition in which a feedback circuit interacts with the internal circuit processes in a way that produces chaotic output behavior.

RE

See **remote environment (RE)**.

Record Level Input/Output (RLIO)

A protocol of IBM Distributed Data Management architecture.

remote component

Components used by a client on a different computer.

remote environment (RE)

A collection of properties that describes a region on the mainframe, or in the case of diagnostic tools such as Capture and Playback, a simulated region. You can view and change these properties by using TI Manager.

remote LU

In an APPC or CPI-C conversation, the logical unit (LU) on the remote end. *Contrast with***local LU**. *See also***remote transaction program**.

remote network address

For an 802.2 connection, a 12-digit hexadecimal address that identifies a remote host, peer, or downstream system. The Remote Network Address in Host Integration Server corresponds to the VTAM MACADDR= parameter in the PORT definition.

remote node

(1) The node at the other end of a connection. (2) The node that contains the logical unit (LU) at the other end of a session. (3) The node that contains the transaction program (TP) at the other end of a conversation.

remote node ID

One of the types of identifiers that can be used to identify a remote node. The remote node ID is an 8-digit hexadecimal number. The first three digits are called the block number, and correspond to the VTAM parameter IDBLK. The last five digits are called the node number, and correspond to the VTAM parameter IDNUM.

remote procedure call (RPC)

A standard that enables one process to make calls to functions that are executed in another process. The processes can be on the same computer or on different computers in the network.

remote transaction program

In an Advanced Program-to-Program Communications (APPC) or Common Programming Interface for Communications

(CPI-C) conversation, the transaction program (TP) on the remote end. *Contrast with***local TP**. *See also***remote LU**.

remote unit of work (RUW)

(1) The form of SQL distributed processing in which the application is on a system different from the relational database, and a single application server services all remote unit-of-work requests within a single logical unit of work. (2) A unit of work that allows for the remote preparation and execution of SQL statements.

Report Program Generator (RPG)

A column-oriented programming language designed for writing application programs for business data processing. RPG requires that certain information, such as control codes and field names, must be placed into specific columns of the program statements.

Request Unit Interface (RUI)

A basic interface that enables programs to acquire and release control of conventional LUs. The RUI also reads and writes request/response headers (RHs), transmission headers (THs), and request/response unit (RU) data. *Contrast with***Session Level Interface (SLI)**.

request/response unit (RU)

Under SNA, a message that controls the session, data flow, and function management aspects of the SNA protocol.

resource dispenser

A service that synchronizes and manages nondurable resources within a process. This service provides for efficient sharing by COM+ objects. For example, the ODBC resource dispenser manages pools of database connections.

Resource Dispenser Manager

A dynamic-link library (.dll) file that coordinates work among a collection of resource dispensers.

resource manager

A system service that manages durable data. Server applications use resource managers to maintain the durable state of the application, such as the record of inventory on hand, pending orders, and accounts receivable. The resource managers work in cooperation with the transaction manager to provide the application with a guarantee of atomicity and isolation (using the two-phase commit protocol). Microsoft® SQL Server™ is an example of a resource manager.

Response Time Monitor (RTM)

A 3270 and NetView facility that monitors the amount of time it takes for a host to respond during 3270 display sessions.

RLIO

See **Record Level Input/Output (RLIO)**.

role

A symbolic name that defines a class of users for a set of components. Each role defines which users are allowed to invoke interfaces on a component.

root

The topmost node in a directory structure.

root directory

The first directory on a drive in which all other files and subdirectories exist.

RPC

See **remote procedure call (RPC)**.

RPG

See **Report Program Generator (RPG)**.

RTM

See **Response Time Monitor (RTM)**.

RU

See **request/response unit (RU)**.

RUW

See **remote unit of work (RUW)**.

-S-

SAA

See **Systems Application Architecture (SAA)**.

safe reference

A reference to the current object that is safe to pass outside the context of the current object.

SAP address

See **service access point (SAP) address**.

SC

See **session control (SC)**.

schema

The definition of the structure of an XML file. A schema contains property information as it pertains to the records and fields within the structure. *See also***document type definition (DTD)**.

SDLC

See **Synchronous Data Link Control (SDLC)**.

security ID (SID)

A unique name that identifies a logged-on user to the security system. SIDs can identify one user or a group of users.

security key

An identifier used by two APPC logical units (LUs) to validate security when a session is activated. The security key performs a function similar to that of a password, but at the LU-LU session level rather than at the TP-conversation level.

security log

The location in which events related to security are recorded when auditing is set up for such events. For example, auditing can be set up to create a security log entry every time the configuration file is changed on a server. *See also***event log**.

security password

The password that is required, along with the security user ID, to gain access to an invoked program when using conversation security.

security user ID

The user ID (also known as user name) that is required, along with the security password, to gain access to an invoked program when using conversation security.

semaphore

A flag variable that is used to govern access to shared system resources.

server

(1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, or a mail server. (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, or a mail server.

server process

A process that hosts COM+ application components in Windows 2000 or later. For example, to use TI, you can drop a TI component (type library) into a COM+ application to create an Automation server that a client application can call. When a client application calls a method on the TI Automation server, the Windows 2000 or later run-time environment loads the TI Automation server along with the TI run-time environment into a surrogate server process that automates the mainframe transaction and passes the results back to the client application.

service access point (SAP) address

A value that codes for access to certain services on an 802.2 connection within an SNA network. The Remote SAP Address

parameter is used for 802.2 connections in Host Integration Server, and corresponds to the VTAM parameter called SAPADDR= in the PU definition.

service TP

A transaction program (TP) that uses APPC to perform services related to SNA functionality. *See also***application TP**; **transaction program (TP)**.

session

(1) A period of time when a connection is active and communication can take place. (2) A set of resources that when activated allow communication to take place. (3) In network architecture, for the purpose of data communication between functional units, all the activities that take place during the establishment, maintenance, and release of the connection. (4) A logical connection between two network-accessible units (NAUs) that can be activated, tailored to provide various protocols, and deactivated as requested. Each session is uniquely identified in a transmission header (TH) accompanying any transmissions exchanged during the session. *See also***LU-LU session**.

session control (SC)

A subcomponent of a transmission control component of a half-session, responsible for activating and deactivating the session and data flow and for receiving the data flow following an error.

Session Level Interface (SLI)

A higher-level interface that facilitates the opening and closing of SNA sessions with host LU 0, LU 1, LU 2, and LU 3 application programs. The SLI permits application programs to control the data traffic at a logical message level. *Contrast with***Request Unit Interface (RUI)**.

session limit

The maximum number of parallel sessions that can be active between two APPC LUs. When an LU-LU session is established, the session limit is negotiated between the two LUs.

severity level

A number that indicates the severity of an audit or error message. Audit messages provide information and have severity 6, 8, or 10. Error messages have severity 12 or 16, indicating a problem that needs to be corrected.

shared assembly

An assembly that can be referenced by more than one application. An assembly must be explicitly built to be shared by giving it a cryptographically strong name. *See also***assembly**; **private assembly**.

SID

See **security ID (SID)**.

side information table

In CPI-C, a table that stores the initialization information required for two programs to communicate. The table resides in the operating systems memory and the system administrator maintains it by accessing a symbolic destination name. The table is derived from the configuration file for Host Integration Server.

single session

A limit of one session between a pair of Advanced Program-to-Program Communications (APPC) logical units (LUs), which limits the associated transaction programs (TPs) to one operation at a time.

SLI

See **Session Level Interface (SLI)**.

SNA

See **Systems Network Architecture (SNA)**

SNA service TP

A transaction program (TP) that uses APPC to perform services related to SNA functionality.

SNA subdomain

With SNA Server version 2.11 and SNA Server version 3.0 or later, you can have several SNA subdomains in a Windows 2000 Server or later domain.

A Windows 2000 Server or later domain:

- Can contain several SNA subdomains.
- Can contain several primary servers, provided that each one is set up in its own subdomain.

With regard to Host Integration Server, each subdomain:

- Contains one primary server.
- Can contain up to 14 backup servers.
- Cannot contain computers running Host Integration Server from other Windows 2000 Server or later domains.

Host Integration Server Setup requires you to specify the name of the subdomain to which the server will belong. One of the SNA subdomains can have the same name as that of the Windows 2000 Server or later domain in which all the servers operate.

Because each subdomain can have only one primary server, it is not advisable to implement an SNA subdomain across slow bridges or routers. Multiple servers in a single subdomain can produce unwanted traffic on the wide-area network.

SnaBase

The SNA Workstation Process. It is present at all times on personal computers whose users want to participate in the SNA network and on personal computers where dynamic loading is to be performed.

SNALink

Link support software that integrates hardware components into a Host Integration Server system. An SNALink is defined when a Host Integration Server system is installed. An SNALink can support only one physical connection from the server.

source TP Name

The host system attempts to identify the source of a request for monitoring, reporting, and so on. The source must be a TP name. MSTX is the default, because it is usually an MS Transaction Server process.

SSCP

See **system services control point (SSCP)**.

string expression

Any expression that evaluates to a sequence of contiguous characters.

stub

An interface-specific object that provides the parameter marshaling and communication required for an application object to receive calls from a client that is running in a different execution environment, such as on a different thread or in another process. The stub is located with the application object and communicates with a corresponding proxy that is located with the client that calls it. In the case of TI, the TI run-time environment serves as the proxy.

subdirectory

A directory contained within another directory in a file system hierarchy.

subdomain

A collection of computers running Host Integration Server that share a single configuration. A subdomain contains one primary server and can also contain one or more backup servers. All servers in a subdomain must belong to the same Windows domain. *See also***backup server**; **primary server**.

SVC

See **switched virtual circuit (SVC)**.

switched SDLC line

A standard telephone line used for SDLC connections on an SNA network. The line is dialed in one of three ways: manually, by a modem that stores the phone number, or by a modem that accepts a phone number string from the software.

switched virtual circuit (SVC)

A type of circuit used by an X.25 connection, in which the circuit is not constantly active, but is called and cleared dynamically. The destination address is supplied when the circuit is called.

Synchronous Data Link Control (SDLC)

A type of link service used for managing synchronous data transfer over standard telephone lines (switched lines) or leased lines.

synchronous transmission

Transmission in which the data characters and bits are transmitted at a fixed rate, with the transmitter and receiver being synchronized. This eliminates the need for individual start and stop bits surrounding each byte. Both SDLC and X.25 use synchronous transmission.

synchronous verb completion

Processing of an SNA verb where the operation of the program is blocked until processing completes. *Contrast with***asynchronous verb completion**.

system administrator

A person who configures, maintains the configuration of, helps users diagnose problems with, and manages a computer system. With Host Integration Server, this person can also be the LAN administrator or a TI developer.

system services control point (SSCP)

(1) A host system network component that provides network services for dependent nodes. (2) An SNA network component that helps control and maintain communication flow between PUs and LUs on the network. Multiple SSCPs can work together to coordinate communications.

Systems Application Architecture (SAA)

Guidelines created by IBM to help developers standardize applications so they function in different operating environments with minimal program modification and retraining of users.

Systems Network Architecture (SNA)

The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks

-T-

TCP/IP

See **Transmission Control Protocol/Internet Protocol (TCP/IP)**.

terminal

A device that can send or receive data over a data communications channel. Host Integration Server includes emulation of 3278 and 3279 terminals.

TH

See **transmission header (TH)**.

thread

The basic entity to which the operating system allocates CPU time. A thread can execute any part of the application's code, including a part currently being executed by another thread. All threads of a process share the virtual address space, global variables, and operating system resources of the process.

TI

See **Transaction Integrator (TI)**.

Token Ring

A type of LAN using the 802.2 protocol, in which a token is passed in a ring around the network, permitting a computer on the network to transmit data only when that computer has the token.

TP

See **transaction program (TP)**.

trace file

A file containing records of internal activities on the SNA network, including calls made to APIs, the activities of APIs, and the activities of communication links and internal flows.

trace message

A message that includes the current status of various COM+ activities, such as startup and shutdown.

tracing

The action of tracking the activities of an application programming interface (API), communication links, and internal flows, including the calls made to APIs. Tracing stores a history of activity in trace files.

transaction

Data entered into a system (such as a customer deposit to a bank account) triggering a certain action (such as updating an account balance).

An atomic unit of work in COM-based systems that either succeeds or fails as a whole or a section of a mainframe COBOL transaction program (TP). A mainframe-based transaction is a section of COBOL code within a transaction program (TP) that completes a certain task or set of tasks.

A mainframe transaction may or may not be an ACID (atomic, consistent, isolated, and durable) transaction. A mainframe-based TP is the actual COBOL program file that contains one or more transactions (sections of COBOL code). A Windows-based transaction is always an ACID transaction that is coordinated by the Microsoft Distributed Transaction Coordinator (DTC).

Data entered into a system (such as a customer deposit to a bank account) triggers a certain action or set of actions (such as updating an account balance) that must all occur or that must all not occur; that is, they act as a unit. That unit is called a transaction in Windows-based terminology.

Each method in a TI component invokes a single mainframe transaction in a mainframe TP. After being invoked, a mainframe transaction can call other transactions in the same or in a different TP.

transaction context

An object used to allow a client to dynamically include one or more objects in one transaction.

transaction ID

The identifier used to invoke a particular CICS or IMS application (transaction program); in CICS, it is the name of the transaction. A transaction ID (TRANID) can be up to four characters in length. The acceptable characters are A-Z, a-z, 0-9, dollar sign ($), at sign (@), period (.), slash mark (/), hyphen (-), underscore (_), percent sign (%), ampersand (&), question mark (?), exclamation point (!), colon (:), vertical bar (|), quotation mark ("), equal sign (=), caret (^), comma (,), semicolon (;), less than sign (<), and greater than sign (>).

Transaction Integrator (TI)

A Windows server-based program that enables you to integrate mainframe or midrange computer transaction programs with component-based and .NET Framework applications.

transaction manager

The transaction manager creates transaction objects and manages their atomicity and durability. Applications request the creation of a transaction object by calling the transaction manager's **BeginTransaction** method.

transaction program (TP)

A COBOL-based mainframe transaction program file. An application program that uses Advanced Program-to-Program Communications (APPC) to exchange data with another TP on a peer-to-peer basis. Within the context of TI, a TP is the mainframe-based CICS or IMS program file that a TI Automation server automates. A TP can contain one or more transactions that are managed on the mainframe side. Each method in a single TI Automation server invokes a single TP. That TP then uses the information passed into it by the TI Automation server to determine which mainframe transaction to run within the TP. Each mainframe transaction within a TP can call other transactions. It is dependent upon how the mainframe COBOL application developer has designed the system.

(1) An application program that uses APPC or CPI-C to exchange data with another TP on a peer-to-peer basis. (2) A program that processes transactions in an SNA network. There are two kinds of transaction programs: application transaction programs and service transaction programs. *See also* **conversation**.

Transmission Control Protocol/Internet Protocol (TCP/IP)

The transport protocol in use by many academic, military, scientific, and commercial organizations to provide communication across wide area networks (WANs). TCP/IP provides communication across interconnected networks that include a variety of different operating systems (such as VMS, UNIX, and Windows 2000 or later).

transmission header (TH)

A header prefix to a message unit flowing within the Path Control Network (PCN), and containing PCN-specific data about routing, sequencing, blocking, and route pacing.

Twinax

A twinaxial connection to a peer system.

twisted-pair cable

Two paired wires, with each wire twisted two or more times per inch to help cancel out noise.

two-phase commit (2PC)

A protocol that ensures that transactions that apply to more than one server are completed on all the servers or none at all. Two-phase commit is coordinated by the transaction manager and supported by resource managers.

type library

A file (or component within another file) that contains Automation descriptions of exposed objects, properties, and methods. Object library (.olb) files contain type libraries. Type libraries that are shipped as stand-alone files use the file extension .tlb. A TI component is an example of a type library (.tlb file).

-U-

UDA

See **universal data access (UDA)**.

UDT

See **user-defined type (UDT)**.

unbounded

Refers to recordsets or arrays. In TI, the rows in a recordset or the elements in an array are transmitted one at a time. Therefore, the mainframe application program must issue multiple receives or sends until all data is transmitted.

This type of parameter or return value can be defined as unbounded for the **CICS Using LU 6.2** and **IMS Using LU 6.2** models only. The number of rows in a recordset or the number of elements in an array is not determined (that is, bounded) before run time. Unbounded parameters or return values can occur anywhere in the Automation method. However, a parameter or return value of this type is always transmitted to and from the mainframe after all other data. TI supports, at most, a single unbounded input parameter and a single unbounded output parameter or a single unbounded in/out parameter.

universal data access (UDA)

A Microsoft data access method which supplies access to information across the enterprise. Universal data access provides high-performance access to a variety of information sources, including relational and non-relational, and an easy to use programming interface that is tool and language independent.

user alert

A message sent by a 3270 user to a host system operator by way of NetView, requesting an action such as mounting a tape or changing forms on a printer.

user identifier

A string of characters that uniquely identifies a user to a system.

user name

The name (also known as user ID) that identifies a Windows user account.

user-defined type (UDT)

A data type that is defined in a program. User-defined data types generally contain many different data types that are defined by the programming language being used. In COBOL, UDTs are called RECORDS (that is, any declaration containing lower-level numbers).

-V-

variable-length string

A fundamental data type that holds character information. A String variable can contain approximately 65,535 bytes (64 KB), and it is either fixed-length or variable-length. Strings usually have one character per byte; however, TI supports Unicode BSTR strings that occupy 16 bits per character. Fixed-length strings are declared to be a specific length, and variable-length strings can be any length up to 64 KB, less a small amount of storage overhead.

VCB

See **verb control block (VCB)**.

verb

Command from one LU to another to exchange data and perform tasks. *See also***APPC verb**.

verb control block (VCB)

A structure made up of variables, which identifies the verb to be executed, supplies information to be used by the verb, and contains information returned by the verb when execution is complete.

VINES

See **VIrtual NEtworking System (VINES)**.

VIrtual NEtworking System (VINES)

A collection of networking software products from Banyan Systems, Inc. VINES includes an addressing system called StreetTalk.

Virtual Telecommunications Access Method (VTAM)

A set of IBM mainframe programs that control communications between mainframe applications and the terminals and computers that connect to the mainframe.

VTAM

See **Virtual Telecommunications Access Method (VTAM)**.

-W-

WAN

See **wide area network (WAN)**.

wide area network (WAN)

A high-speed communication system, consisting of hardware (computers and peripherals) and software (programs and files), that provides communications services and enables resources to be shared over a larger geographic area than that served by a LAN. *Contrast with***local area network (LAN)**.

wildcard character

Synonym for pattern-matching character.

Windows-initiated processing (WIP)

A Windows server platform can access and integrate its programs with the programs on a non-Microsoft server platform (usually a mainframe or mid-range computer such as the AS/400).

WIP

See **Windows-initiated processing (WIP)**.

-X-

X.25

The CCITT standard used for communication over a packet-switching network. X.25 uses the protocol called qualified logical link control (QLLC).

XID

See **exchange identification (XID)**.

XML

See **Extensible Markup Language (XML)**.

XML Schema Definition (XSD)

A language proposed by the W3C XML Schema Working Group for use in defining schemas. Schemas are useful for enforcing structure and constraining the types of data that can be used validly within other XML documents. Unlike DTD, which requires its own language and syntax, XSD uses XML syntax for its language. XSD closely resembles and extends the capabilities of XDR. The W3C now recommends the use of XSD as a standard for defining XML schemas.

XSD

See **XML Schema Definition (XSD)**.

XSL

See **Extensible Stylesheet Language (XSL)**.

-Y-

No terms.

-Z-

No terms.

UI Help

This section contains the user interface (UI) Help that appears when you press the F1 key or click **Help** in the Host Integration Server or Microsoft Visual Studio UI.

In This Section

Installation Help

Use the topics in this section to navigate through the setup user interface.

In This Section

Welcome Screen

Be sure to read the **Welcome** screen because it contains important information.

Setup cannot install system files or update shared files if the files are in use. Before continuing, close all open applications.

Important
Stopping existing SNA Server or Microsoft Host Integration Server services will immediately disconnect users from the Host computer and terminates all sessions.

Viewing the Welcome Screen

1. Click **Cancel** if you would like to quit Setup.

   -or-

   Click **Next** to continue setup.

Note
You can click **Cancel** at any screen to quit the Setup program. If you need to make changes to information you have entered in a previous screen, you can click **Back**.

License Agreement Screen

Read the End-User License Agreement to become familiar with the terms of the Host Integration Server license.

Note
You must accept the License Agreement to complete the server installation.

To accept the End-User License Agreement

1. Select **I accept the License Agreement**.
2. Click **Next**.

User Information Screen

Enter the Full Name and Organization information.

The Full Name is mandatory and the Organization is optional.

To enter the user information

1. Enter the **Full Name** and **Organization** information.
2. Click **Next**.

Select Features Screen

From the **Select Features** dialog box, you can select from several options.

- **Browse** allows you to select the location where Host Integration Server will be installed.
- **Disk Cost** displays the disk space requirements for installed components.
- **Reset** sets all features back to the default setting of all components getting installed.
- **Make Features Unavailable** removes features from your current installation option.

To accept features

1. Either accept all of the Host Integration Server features or make options that you do not want to install unavailable.
2. Click **Next** to continue.

Folder Installation Location

For new installations, Setup displays the current location where Host Integration Server software will be installed. The default location is **C:\Program Files\Host Integration Server**.

For upgrades, the existing folder where the Host Integration Server or a previous version of SNA Server is installed will be displayed.

Current folder location

1. To change the current folder location where Host Integration Server will be installed, click **Browse**.
2. You will be prompted to enter the new folder name. Click **OK**.

Checking Disk Cost

To view the disk cost for installed components, click **Disk Cost** on the **Select Features** dialog box.

This dialog box shows the current disk volumes with the disk size, amount of available space, the amount of space required for selected components, and the difference available.

To check disk cost

1. View the Disk Cost information.
2. Click **OK** to close the **Disk Cost** dialog box.

Resetting Installation Features

You can reset the selected features back to the original settings by clicking **Reset** on the Select Features dialog box.

To reset the install features

1. Click **Reset** to verify that you want Setup to reset all features to be installed.

All features that were marked as unavailable will now be available to Setup.

Make Features Unavailable

To make a Host Integration Server feature unavailable, click the feature icon. One of the two menu options will appear, depending on the item you are selecting. If you make all of the features under **SNA Application Support** unavailable to Setup, the basic SNA Service (SNABASE.EXE) will still be installed. If you make the parent item, SNA Service unavailable, all services under SNA Service will be unavailable to Setup.

The choices for Setup include:

- Will be installed on local hard disk
- Entire feature will be installed on local hard disk

- Entire feature will be unavailable

Note
You will see a red "X" when a feature is marked unavailable.

To make a feature unavailable

1. Click the Host Integration Server feature that you want to make unavailable.
2. Select the **Install** option.

Services Account Screen

Setup installs a number of services that must be able to log on to the Windows 2000 domain. This requires Setup to create a domain user account and to assign the account the privileges that the services require to operate properly.

You must provide a domain and user name (domain/user) along with the password for the account. If the account does not exist, Setup will create it.

Services Account Configuration screen

1. Type the domain and user name (domain/user).
2. Type and confirm the password.
3. Click **Next**.

Begin Installation Screen

All required information for the installation process to start has been entered.

Setup will copy all required files and make all configurations to your system.

Note
If you need to make any changes to information you have entered, click Back and make the appropriate changes.

To start the installation and configuration process

1. Click **Next** to start the Setup process.
2. Setup will now copy all required files to your hard disk.

Finish Installation Screen

To finish the installation and configuration process

1. If you installed components that require SNA support, the Configuration Wizard will start when you click **Finish**.
2. After Setup is completed, click **Finish** to complete the installation process.

Configuration Wizard Help

Use these topics to navigate through the Configuration Wizard.

This section contains:

Common Settings Page

Enable the Common Settings for this HIS Group

Select to configure the settings in the **Security Group** field.

Security Group

Click **Edit** to edit the **HIS Runtime Users** and **HIS Administrators** groups.

Enable Support for 3270, APPC and LUA Applications

Select to configure the settings in the **Resource Location** field.

(Only displayed when the SNA Gateway is not installed.)

Resource Location

Select **Sponsor Server Support** and enter the **Sponsor Server** name, or select **Active Directory Support** and enter the **Organizational** name.

(Only displayed when the SNA Gateway is not installed.)

Advanced

Click to configure advanced settings.

(Only displayed when the SNA Gateway is not installed.)

Windows Service

View or change the current Windows Service accounts.

(Only displayed when the SNA Gateway is not installed.)

Advanced Client Page

Configure the following settings as desired:

- Terminate Sponsor
- Timeout
- Update Random Sponsor list Dynamically
- Select Random Sponsors
- Accept Backup Sponsors
- Run resource location component as an application
- Allow per user settings
- Use the Credentials of
- Logged on User
- This account

Advanced Client Configuration Page

Configure the following settings as desired:

- Terminate Sponsor
- Timeout
- Update Random Sponsor list Dynamically
- Select Random Sponsors
- Accept Backup Sponsors
- Run resource location component as an application
- Allow per user settings
- Use the Credentials of
- Logged on User
- This account

Network Integration Page

Use this page to configure Network Integration.

Enable the Network Integration for this HIS Group

You must select this to make any changes on this page.

Subdomain

If you select the role of **Primary Configuration Server**, enter the **Subdomain Name**.

If you select the role of **Secondary Configuration Server**, enter the **Primary Server Name** and also the **Subdomain Name**.

Services

Select the services you want to have configured.

Advanced

Click to configure Client Protocol Support and Active Directory Support.

Windows Service

View or change the current Windows Service accounts.

Network Integration Advanced Page

Client Protocol Support

Select **TCP/IP**, **Microsoft Networking (Named Pipes)**, or both.

Active Directory Support

Select **Support Active Directory Clients** to make changes in this field. Then enter a valid **Active Directory Name** and **Organizational Unit**.

Data Integration Page

Select **Enable Data Integration Feature** to make configuration changes on this page.

Enable DB2 Distributed Transactions

Select to configure DB2 Distributed Transactions properties, or accept the defaults.

Enable Host Files component (DDM) as a service

Select as desired.

Windows Service

View or change the current Windows Service accounts.

Transaction Integrator Page

Enable Host Initiated Processing

Select this option to make changes on this page.

Then specify a **Server Name** and **Database Name** for the **Database Store**.

Session Integrator Page

Select **Enable Session Integrator Feature**, and then view or change the current **Windows Service** accounts as desired.

MSMQ-MQSeries Bridge Page

Use this page to configure the MSMQ-MQSeries Bridge.

Start Page

Choose either **Basic** or **Custom** configuration.

Basic Configuration

Recommended for first-time users. This configures your system with default settings.

In the **Service Credential** field, enter a **Username** and **Password**.

In the **Network Integration** field, select either **Primary** or **Secondary Configuration Server**.

In the **Database** field, enter the **Database Server Name**.

In the **Data Integration** field, enter the **Partner DB2 Resource Managers** (optional).

Click **Configure** to begin the configuration process.

Custom Configuration

Recommended only for advanced users. This allows you to enable and disable features, and to customize the values for each database and service account.

In the **Service Credential** field, enter a **Username** and **Password**.

In the **Network Integration** field, select either **Primary** or **Secondary Configuration Server**.

In the **Database** field, enter the **Database Server Name**.

In the **Data Integration** field, enter the **Partner DB2 Resource Managers** (optional).

Click **Configure** to begin the configuration process.

Overview Page

The **Overview** page displays the features you have selected to configure, and the status of each.

Click a feature on the left side of the pane to begin configuration.

Service Accounts Page

Use this page to edit the selected service account.

User Name

Type the user name for this service account.

Password

Type the password for this account.

Database Accounts Page

Use this page to edit the server name and database name for the selected feature.

Database server name

Type the name of the database server that will host this feature.

Database name

Type the name of the database for this feature.

Service Accounts View

Use this page to view a list of features and the accounts used to run the services used in your configuration.

Edit

Edit the service account name of the selected features.

You can select multiple features to edit by pressing CTRL and clicking the mouse.

Database View Page

Use the **Consolidated Databases View** page to view a list of features, servers, databases, and data stores used in your configuration.

Edit

Edit the database server name and database name of the selected features.

You can select multiple features to edit by pressing CTRL and clicking the mouse.

Summary Page

Before beginning the configuration process, you can review the selections you have made. Click **Back** to make any changes.

Progress Page

Displays the progress of your configuration.

Finish Page

Review your errors or log file if necessary, and then click **Finish**.

Unconfigure Page

Select the features you want to unconfigure on this computer, and then click **OK**.

SNA Manager Help

Use these topics to navigate through the SNA Manager user interface.

In This Section

Active TN5250 Sessions

AS/400 Definition Properties

APPC Modes Folder

CPI-C Symbolic Names Folder

Link Service Adapter

Link service allows Host Integration Server to communicate with host, peer, or downstream computers over token ring and Ethernet Local Area Networks.

Configure a DLC 802.2 Link Service

A DLC 802.2 link service allows Host Integration Server to communicate with host, peer, or downstream computers over token ring and Ethernet Local Area Networks.

Title

The default title is DLC 802.2 Link Service #1. To change the default title, delete the title and enter a new title for this link service. You can use up to 40 characters for the title.

Adapter

Select the adapter card you are using for this service.

Local Service Access Point (SAP)

Enter the local SAP. This number must be a multiple of 4, ranging from 4 through 252. In most cases, the SAP for Host Integration Server is 4.

Select **Use Fixed SAP** to use a fixed SAP address.

Select **Allow Link Service to be Distributed** to allow this service to be distributed.

Demo SDLC Link Service

Configure a demonstration SDLC link service. This service uses a script file to simulate an SDLC link service.

Title
The default title is Demo SDLC Link Service #1. To change the default title, delete the title and enter a new title for this link service. You can use up to 40 characters for the title.

Script File
Select a demonstration script file from the drop-down list.

Select Allow Link Service to be Distributed to allow this service to be distributed.

Distributed Link Service Properties

Service Title

The default title is Distributed Link Service #1. To change the default title, delete the title and enter a new title for this link service. You can use up to 40 characters for the title.

Link Type

Select the type of link service represented by this Distributed Link Service from the drop-down list.

Primary Remote Link Services

Enter the name of the remote link service used by this service.

Alternate Remote Link Services

Enter the name of an alternate remote link service used by this service.

Select **Compression used** if data compression is to be used on this link service.

APPC Mode Properties

The following tabs are available on the APPC Mode Properties Sheet:

APPC Mode Properties: General

Mode Name

Enter a Mode Name.

Comment

Optionally, type a comment of not more than 25 characters.

APPC Mode Properties: Limits

Parallel Session Limit

Specify the maximum number of sessions allowed with this mode. To allow only a single session, specify 1. To allow parallel sessions, specify a value greater than 1. If the local APPC LU to be used with this mode is dependent, specify 1; dependent local APPC LUs cannot have parallel sessions. One of the key mode properties is the **Parallel Session Limit**. This limit determines whether an LU-LU pair can carry on only one interaction at a time (a parallel session limit of 1), or multiple concurrent interactions (a parallel session limit greater than 1). If parallel sessions are to be allowed with an LU-LU pair, the local LU must be independent, and the remote LU in the pair must support parallel sessions.

If the LU-LU pair can carry on multiple parallel sessions, other mode properties, such as minimum contention winner limit, determine to what extent each LU can initiate interactions.

Minimum Contention Winner Limit

Specify the Minimum Contention Winner Limit. The sum of the Minimum Contention Winner Limit and the Partner Min Contention Winner Limit must be less than or equal to the Parallel Session Limit.

Partner Min Contention Winner Limit

Specify the Partner Min Contention Winner Limit. The sum of the Partner Min Contention Winner Limit and the Minimum Contention Winner Limit must be less than or equal to the Parallel Session Limit.

Automatic Activation Limit

Specify the number of contention winner sessions to be activated for the local LU whenever the connection for this mode is started. In a local contention winner session, the local LU can initiate conversations without permission from the partner LU.

This does not apply for single-system APPC (communication between two local LUs on the same system).

APPC Mode Properties: Characteristics

Pacing Send Count

Specify the maximum number of frames for the local LU to send without an SNA pacing response from the partner LU. If you specify 0, the local LU can send an unlimited number of frames without receiving a response; in this case, the partner LU can negotiate and set a limit on the count.

Pacing Receive Count

Specify the maximum number of frames for the local LU to receive from the partner LU before the local LU sends an SNA pacing response. If you specify 0, the local LU can receive an unlimited number of frames without sending a response.

Max Send RU Size

Specify the maximum size for RUs sent by the TP(s) on the local system.

It is not necessary to set a minimum send RU size, which is 256 on Host Integration Server.

Max Receive RU Size

Specify the maximum size for RUs received from the TP(s) on the remote system.

It is not necessary to set a minimum receive RU size, which is 256 on Host Integration Server.

High-Priority Mode

Select to give preference to communication with this mode over communication with a low-priority mode.

APPC Mode Properties: Partners

This tab allows you to **Add** or **Remove** APPC LU partnerships from the Host Integration Server configuration file. The mode name appears in the title. The list box shows the **Server**, **Local LU Alias**, **Remote LU Alias**, **Connection**, and which LUs are partnered with the APPC mode in the title.

Remove

Click **Remove** to delete APPC LU partnerships from the Host Integration Server configuration file.

Add

1. Click **Add** to add APPC LU partnerships from the Host Integration Server configuration file. The **Add Partnerships** dialog box appears.
2. Select a **Server name** from the subdomain by clicking the DOWN arrow in the drop down list.
3. After the server is selected, a list box showing **Local LU**s and a list box showing **Partner LU** and **Connection** are filled. Make selections from each list to establish partnerships.
4. Click **Apply** to add pairs to the **APPC Mode Properties: Partners** page while keeping the **Add Partnerships** dialog active.
5. When you finish selecting partnerships, click **OK** on the **Add Partnerships** dialog to return to the **APPC Mode Properties: Partners** page.
6. Click **OK** on the **APPC Mode Properties: Partners** page to commit the changes to the Host Integration Server configuration file.

Note
You can select multiple modes for partnerships using the same LUs. Press the SHIFT key and the CTRL key and click to select non-contiguous modes.

Note
The Host Integration Server configuration file is updated dynamically. There is no need to stop and restart SNA service.

APPC Mode Properties: Compression

Maximum Send Compression

Select the maximum compression desired from the drop down list. The valid values are: **None**, **Run Length Encoding (RLE)**, and **LZ9**. These options offer progressively better compression, but at a progressively higher CPU usage.

Maximum Receive Compression

Select the maximum compression desired from the drop down list. The valid values are: **None**, **Run Length Encoding (RLE)**, and **LZ9**. As with Maximum Send Compression, these options offer progressively better compression, but at a progressively higher CPU usage cost.

Allow LZ and RLE Compression

If LZ9 is used, this option controls whether data is compressed using RLE before being further compressed using LZ9.

Endpoint Only Compression

This option controls whether intermediate nodes may use compression if one of the endpoints does not support or otherwise does not use compression.

3270 LU Properties: General

LU Number
Enter the LU Number for LUs on 802.2, SDLC, or X.25.

Enter an appropriate LU Number.

LU Name

Type the LU Name.

Connection

The connection for this LU is shown. The connection cannot be changed from this dialog box.

Pool

If the LU has already been assigned to a pool, the pool name appears here.

Comment

Optionally, enter a comment of not more than 25 characters.

Use Compression

Select this option to enable 3270 LU data stream compression. This option must also be set in the host VTAM configuration for the LU.

User Workstation Secured

Select this option to enable a higher level of security. When the option is selected the user can only acquire an LU if:

- the user's User Record is assigned to this LU, and
- the user's workstation has been defined with a Workstation Definition.

3270 Display LU Properties: Associated Printer LU
Display LU

Shows the display LU name if you have already created a display LU; otherwise, this will be filled in when you complete the configuration of this display LU.

Associated Printer LU

Click the DOWN arrow to display the list of available printer LUs on this connection. Choose a printer LU to associate with this display LU. If you have not created any printer LUs on this connection, you must do so before they will appear in the list.

3270 LU Properties: LUA

The following tabs are available on the 3270 LUA Properties Sheet:

3270 LU Properties: LUA General Tab

LU Number

Enter the LU Number for LUs on 802.2, SDLC, or X.25.

Enter an appropriate LU Number.

LU Name

Type the LU Name.

Connection

The connection for this LU is shown. The connection cannot be changed from this dialog box.

Pool

If the LU has already been assigned to a pool, the pool name appears here.

Comment

Optionally, enter a comment of not more than 25 characters.

Use Compression

Select this option to enable 3270 LU data stream compression. This option must also be set in the host VTAM configuration for the LU.

User Workstation Secured

Select this option to enable a higher level of security. When the option is selected the user can only acquire an LU if:

the user's User Record is assigned to this LU, and

the user's workstation has been defined with a Workstation Definition.

High Priority Mode

Select this option to increase the priority for this LU.

Note
Two additional property pages appear when the LU is assigned to a TN3270. These are the TN3270 property page and the IP Address List page.

3270 LU Properties: TN3270 Tab

Type

Select from the list the display or printer type.

Generic Display

This is the default pool type. The display LU or pool will be assigned to a client computer that either does not specify a particular LU or Pool, or that requests this LU or Pool by name.

Specific Display

The LU or pool configured as specific will only be assigned to client systems that request this LU or pool, and not to clients making generic requests.

Generic Printer

The printer LU or pool will be assigned to a client computer that either does not specify a particular LU or pool, or that requests this LU or pool by name. When this option is checked, the **terminal name** will default to IBM-3287-1.

Specific Printer

The printer LU or pool configured as specific will only be assigned to client computers that request this LU or pool, and not to client computers making generic requests. When this option is checked, the **terminal name** will default to IBM-3287-1.

Associated Printer

Printer LUs can be marked as associated instead of generic or specific. To associate a printer LU with a terminal LU, select this option and choose the **Associated LU** from the drop-down list box. When this option is checked, the **terminal name** will default to IBM-3287-1.

Sessions

This is the number of TN3270 sessions allowed for the selected LU or pool. The number of TN3270 sessions must not exceed the number of LUs listed under Host Integration Server Resource Information.

If you reduce the TN3270 Sessions limit to 0 (zero) TN3270 service will not assign the LU or pool.

Associated Printer

When the **Type** option **Associated Printer** is selected, choose an **Associated Printer** from the drop-down list. In this case, the printer LU is partnered with one terminal LU. A client computer accesses that printer LU by sending in an "associate" request and giving the name of the terminal LU with which the printer LU is partnered.

Terminal types

TN3270 service will default to IBM Terminal Model 2. To add a different terminal model to this LU, click the corresponding number (**2**, **3**, **4**, or **5**) to the left of the list of terminal names. After a Terminal Model is selected, Terminal Names are automatically assigned. You can remove selection of individual terminal names that do not apply to your network configuration by clearing the checkbox opposite the name, or you can clear all terminal names by clicking **Clear**.

Note
The terminal name strings selected must be compatible with the host logmode entry for the LU. For printer LUs, the **Terminal Name** will default to IBM-3287-1.

Port

Select **Default port** to use the port number configured with the service (on the Service Properties page), or select a port from the list. The list displays all of the defined TN3270 ports along with their Security setting (High, Medium, Low, None, or Unsecured).

Note
TN services listen on multiple ports simultaneously. You can set a default number for the TN service and override this number on a per session basis, allowing a single client computer to connect to multiple host computers.

Note
TN3270 client systems can only access LUs that are configured as generic terminal LUs. TN3270 client systems can access generic, specific, and associated LUs.

Note
Configuration changes are apparent only to users who establish a connection after the configuration changes are saved. Users who were connected at the time the configuration changes were made will not be affected.

3270 LU Properties: IP Address List

You can associate this LUA to a specific IP address or server name.

Add Address

Click to specify the **IP Address** and **Subnet Mask** of the client workstation to which you are granting access.

IP Address

Click an entry in the **IP Address** column to edit the IP address for a client workstation.

Subnet Mask

Click an entry in the **Subnet Mask** column to edit the subnet mask for a client workstation.

Add Name

Click to add a workstation name to the **IP Address** list box. The workstation name identifies that a client workstation whose

Click to add a workstation name to the **IP Address** list box. The workstation name identifies that a client workstation whose address resolves to that name can connect to the service.

Remove

Click to remove an entry that is currently highlighted in the **IP Address** list box.

Clear All

Click to remove all entries in the **IP Address** list box. The service will then be configured to allow access from any client workstation.

Note
The IP address must be changed from the default value of 0.0.0.0 if you want to assign a specific IP address. If you try to add t he default IP address with the default subnet mask value, you will receive a message indicating an Invalid IP Address/Subnet Mask.

Note
The IP address of an incoming connection is compared to each of the available LUs or pools in turn. Each resource has an ass ociated list of IP address patterns and subnet masks. The IP address from the incoming connection is masked (by bit and by t he resource's subnet mask), then compared to the resource's IP address pattern masked by the resource's subnet mask. If th e result is equal, the connection is allocated to this resource.

3270 LU Properties: Down Stream

Enter the LU Number for LUs on 802.2, SDLC, or X.25.

Enter an appropriate LU Number.

LU Name

Type the LU Name.

Connection

The connection for this LU is shown. The connection cannot be changed from this dialog box.

Pool

If the LU has already been assigned to a pool, the pool name appears here.

Comment

Optionally, enter a comment of not more than 25 characters.

Pool Properties: General

Pool Name

Enter a name for this pool.

Comment

Optionally, enter a comment of not more than 25 characters.

Pool contains Display LUs with Associated Printers

This option is intended for users whose host applications have a direct relationship between display LUs and printer LUs (i.e., this applies to display-type pools only). With this option selected, all display LUs in the pool will be treated as though they have associated printer LUs. Also configure the display and printer LUs with the LU numbers that the host application is expecting.

Pool Properties: Display Model

Display Model

If you select **Display** for the LU Type, you can select a Display Model.

When an LU is assigned to a pool, the display model setting for the pool overwrites the setting of the LU, and the setting displayed in the **3270 LU Properties** dialog box is dimmed.

Some emulators can only emulate certain display models. For more information, see your emulator documentation. If there is a conflict between the Display Model setting for the pool and the Display Model setting for an LU in the pool, a message box appears. You can configure the individual LU setting to change or exclude the LU from the pool.

Model can be overridden

Select this check box to allow the user to override the display model type by using a 3270 terminal emulation program.

User Properties

User Name

Provided for information only. This field cannot be changed here.

Domain

Provided for information only. This field cannot be changed here.

Comment

Optionally, type a comment of up to 25 characters.

Use Client/Server Encryption

Check this box to enable encryption between the client system and Host Integration Server.

Allow Access To Dynamically Created Remote APPC LUs

Check this box to let this user or group use dynamically created APPC LUs.

APPC Defaults

Local APPC LU

If this user will be using APPC programs (TPs, 5250 emulators, or APPC applications), you can choose a default local APPC LU to be used when the user starts such programs.

Avoid assigning a default local APPC LU to the group "Everyone". Instead, to make a local APPC LU available to any user, when configuring the local LU, select the check box labeled **Member of Default Outgoing Local APPC LU Pool** on the **Advanced** tab of **Local LU Properties**.

Remote APPC LU

If this user will be using APPC programs (TPs, 5250 emulators, or APPC applications), you can choose a default remote APPC LU to be used when the user starts such programs.

To accept the settings, click **OK**; to exit the dialog box without accepting the settings, click **Cancel**.

When you assign default APPC LUs to user and group accounts with overlapping memberships, some of these assignments override others.

TN3270 Properties: General

Name

Displays the name of the server running the TN3270 service. This field cannot be edited here.

Comment

Optionally, type a comment of up to 25 characters.

Use Name Resolution

Name Resolution should only be selected if you are running a domain name resolver. A domain name resolver catalogs IP addresses and corresponding network names of connected computers. The domain name resolver allows you to enter the name of a computer rather than the IP address when an IP address is required.

TN3270 Mode only

The TN3270 service also supports TN3270E, an enhancement to TN3270. When a client computer first connects to a computer running TN3270 service it negotiates which functions they both support. TN3270 emulators should be able to negotiate with TN3270 service, if only to state that they do not support TN3270. However, some TN3270 emulators are unable to negotiate properly with TN3270 service, causing the negotiation to fail. For this reason the TN3270 service has an option to default to TN3270 mode and not to use TN3270 features, so that these TN3270 negotiation problems do not occur.

Printer Flow Control

If a TN3270 service adheres strictly to the specification described in RFC 1647, there is no way of implementing flow control between a computer running TN3270 service and a TN3270 client. In practice this causes no problems for display emulators, but it does cause a problem for printer emulators, which can be overloaded with data and have no way of notifying the TN3270 service that they cannot process any more messages. If this option is turned on, the TN3270 service sends all messages to a TN3270 printer client as RESPONSE-REQUIRED, and does not send any messages until it has received a response for the previous message.

Close Listen Socket

By default the TN3270 service always has a socket open to listen for incoming requests. If this option is turned on, the TN3270 service stops listening on this socket after all of its defined LUs are in use. The purpose of this is to work with emulators that can try to connect to a number of computers running TN3270 service and that connect to whichever computer accepts their connection attempt. In this case it is useful if a computer with no LUs available is not listening.

Log Normal Audit Events

If this is set, audit messages are logged. These are messages that log successful client connection and successful client termination.

Use SNA Event Log

If this is selected, all TN3270 service event messages are written to the event log being used by the Host Integration Server system. If this is not set, all TN3270 service event messages are written to the Windows event log on the local machine.

TN3270 Properties: Port/Security

This property page contains two groups.

The **Defined ports** group displays information pertaining to the currently configured 3270 Server ports.

The **Configure ports** group consists of controls for adding, editing, and deleting port configurations.

Defined ports

Port

Lists all ports and corresponding Security settings. On a new installation, setup automatically defines Port 23 as the Default Port.

Security

Displays the level of encryption/authentication assigned to that port. Values are High (168), Medium (128), Low (40), and Unsecured (TLS/SSL not enabled). Default is **Unsecured**.

Client certificate

Displays status: Required or Not required. Default is **Not Required**.

Comment

Displays a comment.

Port configuration

Port

Enter a valid port number from 1-65535.

Security

Choose a setting from the list: High (168), Medium (128), Low (40), or Unsecured. Default is **Unsecured**.

Comment

This is optional. Maximum length allowed is 25 characters.

Verify client certificate

If selected, the client will have to provide a valid certificate to make the connection. Default is not selected. If Security level is set to Unsecured, this option is unavailable.

The client requires a valid Certificate with the following properties:

- Type X509
- Client Authentication
- Associated private key

These certificate settings may match some of those you would not choose to grant access to. It is therefore recommended that you check the list of default Trusted Root Certification Authorities in the TN3270 Service Store, and remove any you do not want to be there.

TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client computer to connect to multiple host computers.

You can also add the TN3270 port by using the SnaCfg tool, with the command and parameters shown below:

```
TN3PORT tn3270Server:PortNumber /ADD
   /COMMENT:"comment"
   /SECURITY:{ None | Low | Medium | High }
   /CLIENTSertVALAUTH:{ Yes | No }
```

The properties of the command are described in the following table:

Property or Method	Description	Validation
/COMMENT:Text	The comment field	0 – 25 characters
/SECURITY:Type	Security level	None, Low, Medium or High
/CLIENTCertVAL:YesNo	Indicates whether the client certificate verification is enabled	No – disabled Yes - enabled

Default Port

On a new installation, setup automatically defines Port 23 as Default Port. There can only be one default port at a time. After a port has been designated Default Port, it cannot be deleted until another Default Port has been selected. The default is **Not Selected**.

Adding Ports and Security

This property page allows increased security through support for Secure Sockets Layer (SSL) and TLS for all network transport-level services.

Although the Microsoft 3270 Client (emulator) does not support SSL or TLS, many third-party software vendors offer 3270 emulators that support this functionality, including Attachmate, IBM, NetManage, and WRQ.

To add a new port

1. On the **TN3270 Properties: Port/Security** page, click **New**.
2. Enter the port number and select a security level (default is Unsecured). You can also add a comment and make this the default TN3270 Server port.
3. For maximum security, check the Client Certificate checkbox.

To edit the properties of an existing port

1. On the **TN3270 Properties: Port/Security** page, highlight the port.
2. Click Edit.
3. When you are finished, click **Save**.

TN3270 Properties: Settings

Idle Timeout

Specify time limits. If the session is inactive for this length of time, then TN3270 service disconnects the client system.

Init Status Delay

Specify time limits. This is the delay between the time when TN3270 service connects to a host session and the time the TN3270 service starts updating the client screen. There are often a large number of startup messages when the TN3270 service first connects to a host session, and this option gives the user the opportunity not to receive them all.

Message Close Delay

Specify time limits. When TN3270 service forces a client computer to disconnect (for example, when the Host Integration Server session to the host has been lost), it sends the client computer an error message to be displayed on the screen. This value specifies the time between sending the message to the client computer and closing the socket with the client computer (which causes some client computers to clear the screen, and so erase the message).

Refresh Cycle Time

Specify time limits. This is the delay between updates of the status on the display.

Default RU Sizes - Inbound and Outbound

This controls the RU size (SNA message size) used by the TN3270 service for logon messages to and from the host. The minimum value for inbound or outbound RU size is 256 bytes. If the host application sends large logon screens, these values should be increased.

Certificate CN

The common name of the certificate used if TLS/SSL is enabled.

Connection Properties: SDLC

The following tabs are available on the SDLC Connection Properties sheet:

Connection Properties: General

Name

Enter a Name.

Link Service

Select a link service from the drop down list. If the link service you want is not in the list, click **Cancel**.

Comment

Optionally, enter a comment of not more than 25 characters.

Remote End

Select the type of remote system for this connection:

- Host System
- Peer System
- Downstream
- PU Passthrough

If this connection will be used for 3270 or LUA LUs, then select Host System.

If you will be using dependent APPC, which relies on a host system for communications, then select Host System, not Peer System.

Allowed Directions

Select the allowed call direction(s), **Outgoing Calls** and/or **Incoming Calls**.

Selecting **Outgoing Calls** causes Host Integration Server to initiate the link-level connection to the remote system or host. If a remote system tries to initiate a connection, Host Integration Server will not accept the connection attempt.

Selecting **Incoming Calls** causes Host Integration Server to accept connection attempts from remote systems or hosts.

If both are selected, Host Integration Server will both initiate a connection and accept connection attempts by other systems.

When multiple SDLC connections all use the same link service, and the connections accept incoming calls, the encoding (**NRZ/NRZI**) settings for all the connections must match.

Activation

If **Outgoing Calls** are included in Allowed Directions, select an Activation setting:

On Server Startup. Select this option if you want the connection to be readily available whenever the server is started.

On Demand. Select this option if you want the connection to be started when needed, and stopped when not in use.

By Administrator. Select this option if you want the administrator to control the connection on a case-by-case basis.

Passthrough via Connection

Displays a list of available (unpaired) PU Passthrough Connections. Connections that have already been paired are not available. Choose a connection that will route you to your destination PU. If PU Passthrough is not selected in the Remote End field, this list is unavailable.

Supports Dynamic Remote APPC LU Definition

Automatically configures the APPC Remote LUs as users request them. This feature requires that an APPN End Node or Net Node be available on the connection. This feature is for convenience, and should not be used in situations of restricted access. If Host System or Peer System is not selected in the Remote End field, this option is unavailable.

With dynamic APPC configuration, if a user requests a session with a valid remote LU, the connection will be established even if the remote LU has not been configured in SNA Manager. If a connection is designated to support dynamic APPC configuration, Host Integration Server will automatically define a remote LU and partner it with a local LU when needed. However, to take advantage of session status and other features of Host Integration Server, it is recommended that administrators configure commonly used remote LUs.

Connection Properties: Address

Dial Data

If this connection uses a switched line and the phone number is not stored in the modem, enter the dial data. With a leased SDLC line, this field is unavailable.

For ways to supply a phone number to the modem, see Phone Number Storage and Modem Types below.

Poll Address

Type a two-digit hexadecimal number. Contact the administrator of the remote system to determine the appropriate value. If the remote system is a host, the local poll address should match the VTAM PU definition for the ADDR= parameter.

If the remote system is a peer, the poll address can be any value except the reserved values 00 and FF. When the connection is used, the link roles will be negotiable; the system that takes on the primary role uses the poll address of the other system, so that the poll addresses match during the session.

Note
Do not use 00 or FF. These values are reserved.

Encoding

Select the encoding scheme for your modem to use.

Your modem must use the same encoding scheme as the modem at the remote computer.

One of two encoding methods for modems:

- NRZ — Non-return to zero
- NRZI — Non-return to zero inverted

For connections to host systems, the encoding scheme must match the NRZI setting in the LINE/GROUP definition in VTAM. Obtain this setting from the host administrator. If VTAM does not specify an NRZI setting, it defaults to NRZI=YES.

When multiple SDLC connections all use the same link service, and the connections accept incoming calls, the encoding (**NRZ/NRZI**) settings for all the connections must match.

Phone Number Storage and Modem Types

Host Integration Server can store a phone number and then send the number (in ASCII) to a modem, which dials the number. This requires that the modem be attached to an SDLC adapter with a built-in serial (COM) port (for example, the SDLC adapters from MicroGate).

Several connection parameters must be set to work with your modem: Dial Data the duplex setting, half-duplex or full-duplex, and the encoding scheme.

Changing the Host Integration Server Dialing Method

To change the Host Integration Server dialing method for a modem, use the Host Integration Server Link Service Setup program to modify the link service. Restart SNA Manager and, from the **File** menu, choose **Save Configuration**.

Modem Requirements for Accepting a Phone Number from Host Integration Server

To accept a phone number from Host Integration Server, your modem must be set up so that it will do the following:

- Accept dial commands in ASCII (eight data bits, no parity bit, one stop bit).
- Not dial when the DTR signal is raised.
- Set Clear to Send (CTS) and Data Set Ready (DSR) to **On** when ready to accept dial commands.

- Set DSR to **Off** after accepting a dial command.
- Set DSR to **On** again when (and only when) the dialed connection is made.
- Change to **Synchronous** mode after the dial-up has completed.
- Change back to **Dial-command** mode if Data Terminal Ready (DTR) is dropped and raised again.

Host Integration Server Actions When Sending a Phone Number

When Host Integration Server is sending a phone number to a modem, it ignores modem responses and holds the modem interface signals Select Standby and DTR to **Off**.

The dialing attempt initiated by Host Integration Server is assumed to have failed if one of the following occurs:

- DSR stays on after the dial string has been sent.
- The connection time-out expires before DSR comes on to indicate that the call is connected.

When Host Integration Server creates a dial string to send to a modem, it uses the outgoing command string supplied in Setup as a base, and then appends the Dial Data (phone number) configured in SNA Manager, followed by a carriage-return line-feed sequence.

Affiliate Application

If you selected Single Sign-On, choose an Affiliate Application from the list. The Enterprise Single Sign-On (SSO) Affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line of business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

Connection Properties: System ID

Local Node Name

Network Name

If you are instructed by the administrator of a remote host, peer, or downstream system, and if you are using Format 3 XIDs, type the Network Name of the remote system. Format 3 is the default XID type.

The Network Name works with the Control Point Name to identify a system. If either of these parameters is supplied, the other should also be supplied.

Control Point Name

If you are instructed to by the administrator of a remote host, peer, or downstream system, and if you are using Format 3 XIDs, type the Control Point Name of the remote system. Format 3 is the default XID type.

The Control Point Name works with the Network Name to identify a system. If either of these parameters is supplied, the other should also be supplied.

Local Node ID

If this connection uses a switched SDLC line (standard telephone line) to connect to a host system, type the Local Node ID. Use the same Local Node ID for all connections and link services on a particular server.

If this connection uses a leased SDLC line, you can use the default for the Local Node ID.

Note
Do not use 000 or FFF for the first three digits; these values are reserved.

XID Type

Select the XID Type. Most systems use Format 3.

If you want to use independent APPC LUs on this connection, you must select Format 3.

When Host Integration Server is configured to use Format 0 over a host connection, the host treats it as a PU type 2.0, and so Host Integration Server exercises only its PU type 2.0 capabilities. When Host Integration Server establishes a host or peer connection using Format 3 and independent LUs, Host Integration Server exercises its PU type 2.1 capabilities.

For configuration changes to take effect, restart the server.

Remote Node Name

Network Name (Remote Node)

If you are instructed by the administrator of a remote host, peer, or downstream system, and if you are using Format 3 XIDs, type the Network Name of the remote system. Format 3 is the default XID type.

The Network Name works with the Control Point Name to identify a system. If either of these parameters is supplied, the other should also be supplied.

Control Point Name (Remote Node)

If you are instructed to by the administrator of a remote host, peer, or downstream system, and if you are using Format 3 XIDs, type the Control Point Name of the remote system. Format 3 is the default XID type.

The Control Point Name works with the Network Name to identify a system. If either of these parameters is supplied, the other should also be supplied.

Remote Node ID

If instructed to by the administrator of a remote host, peer, or downstream system, type the Remote Node ID.

Note
Do not use 000 or FFF for the first three digits; these values are reserved.

Peer DLC Role

Select the DLC role for this connection. The options are:

- Primary
- Secondary
- Negotiable

Connection Properties: SDLC

Max BTU Length

Specify the Maximum Basic Transmission Unit (BTU) Length.

The range is from 265 through 16393; the default is **265**.

With an IBM SDLC adapter, set the Max BTU Length to 521 or less.

Maximum Basic Transmission Unit length. The maximum number of bytes that can be transmitted in a single data-link information frame. A BTU is sometimes called an I-frame.

For downstream connections, specify a Max BTU Length less than or equal to the maximum value supported by software on the downstream system.

For host connections, specify a Max BTU Length less than or equal to the VTAM PU definition for the MAXDATA= parameter.

Data Rate

Select the Data Rate.

The data rate for transmissions between the Host Integration Server communications adapter and the modem:

- **Low** gives more reliable transmissions and prevents the transmission errors sometimes caused by poor-quality lines at the high rate. If a fall-back line speed setting is supported by your modem and communications adapter, and you want to enable it, select **Low**.
- **High** gives faster transmissions. If you select **High**, the CCITT line 111 on the V24 interface of your modem is set to on.

Check your adapter and modem manuals to find out if the data rate can or must be set for your equipment.

Duplex

Select the setting that matches your modem:

- For a half-duplex modem, select **Half**. If none of your adapters were installed with the Constant RTS option set, you can only choose half-duplex.

- For a full-duplex modem, select **Full**. If you want to use the full-duplex setting, one or more of your adapters must have the Constant RTS option set (at installation).

Most servers will use the default, **Half**.

Idle Timeout (for host or peer connections)

Specify the length of time, in tenths of a second, for the local system to wait for a response to a transmission before trying again. Too small of a timeout can cause connection problems. This parameter affects sessions in which the server has a link role of secondary; therefore, it affects all sessions with a host system and some sessions with a peer system (where the link role is negotiable).

The range is from 1 (one-tenth of a second) through 300 (30 seconds). The default is **300** (30 seconds).

Idle Retry Limit (for host or peer connections)

Specify the number of times for the local system to try to send data to the remote system if there is no response. This parameter affects sessions in which the server has a link role of secondary; therefore, it affects all sessions with a host system and some sessions with a peer system (where the link role is negotiable).

The range is from 1 through 255; the default is **10**.

Poll Rate

If the remote system is a peer or downstream system, specify the **Poll Rate** (in polls per second).

The range is from 1 through 50 polls per second; the default is **5**.

Poll Timeout (for peer or downstream connections)

Specify the length of time, in tenths of a second, for the local system to wait for a response to a poll before trying again. Too small of a timeout can cause connection problems. This parameter affects sessions in which the server has a link role of primary; therefore, it affects all sessions with a downstream system and some sessions with a peer system (where the link role is negotiable).

The range is from 1 (one-tenth of a second) through 300 (30 seconds). The default is **10** (1 second).

Poll Retry Limit (for peer or downstream connections)

Specify the number of times for the local system to poll the remote system if there is no response. This parameter affects sessions in which the server has a link role of primary, therefore, it affects all sessions with a downstream system and some sessions with a peer system (where the link role is negotiable).

The range is from 1 through 255; the default is **10**.

Select Standby

If you want your modem's Standby line to be set to on, select this check box.

Check your adapter and modem manuals to find out if standby can be set for your equipment.

Multidrop Primary

If this is a leased SDLC line to downstream systems, and this server will be the primary station for a multidrop connection, then select this box.

This is a connection through which one primary computer simultaneously communicates with multiple secondary computers. Host Integration Server supports multidrop connections to peer or downstream systems on leased SDLC lines.

Contact Timeout

Specify the length of time, in tenths of a second, which the SDLC link service waits for an XID or SNRM response before resending the XID or SNRM.

This parameter is ignored for incoming calls.

The range is from 5 (five-tenths of a second) through 300 (30 seconds).

The default for a host connection is 300 (30 seconds). The default for a peer or downstream connection is 10 (1 second).

Contact Retry Limit

Specify the maximum number of times the link service will resend an XID or SNRM before declaring an outage to Host Integration Server.

This parameter is ignored for incoming calls.

The range is from 1 through 20; the default is **10**.

Connection Dialing Timeout

If this connection uses a switched SDLC line (standard telephone line), specify the number of seconds that will be allowed for the user or modem to dial and make a connection to the remote computer. If the dialing will be done manually, be sure to allow enough time for the user to dial, wait for an answer, and make the connection.

The range is from 10 through 500 seconds; the default is **300**.

This parameter is ignored by incoming calls.

To exit the dialog box without accepting the configuration settings, click **Cancel**. To accept the settings, click **OK**.

To put configuration changes into effect, restart the server.

Connection Retry Limits

Supply activation limits for the connection.

Maximum Retries

Select the number of attempts for Host Integration Server to make when trying to establish the connection. After making this number of attempts, Host Integration Server makes an entry in the event log and stops trying. The range is from 1 through No Limit; the default is **No Limit**.

Delay After Failure

Select the length of time for Host Integration Server to wait between attempts to establish the connection. The range is from 5 seconds through 255 seconds; the default is **10** seconds.

Connection Properties: DLC 802.2

The following tabs are available on the DLC 802.2 Connection Properties sheet:

General Tab

The following information must be entered on the **General** tab for the 802.2 connection:

- Connection Name
- Link Service
- Comment (optional)
- Remote End
    - Host System
    - Peer System
    - Downstream
    - PU Passthrough
- Allowed Directions
    - Outgoing Calls
    - Incoming Calls
    - Both Directions
- Activation
    - On Server Startup
    - On Demand
    - By Administrator
- Passthrough via Connection
- Supports Dynamic Remote APPC LU Definition

Address Tab

The following information must be entered on the **Address** tab for the 802.2 connection:

- Remote Network Address
- Remote SAP Address
- Local SAP Address

System Identification Tab

The following information must be entered on the **System** tab for the 802.2 connection:

- Local Node Name
    - Network Name
    - Control Point Name
    - Local Node ID
- XID Type
- Remote Node Name
    - Network Name
    - Control Point Name
    - Remote Node ID
- Peer DLC Role
- Compression Type

DLC 802.2 Tab

The following information must be entered on the **DLC 802.2** tab for the 802.2 connection:

- Max BTU Length
- Receive ACK Threshold (frames)
- Unacknowledged Send Limit (frames)
- Retry Limit
- XID Retries
- 802.2 Timeouts
    - Response (t1)
    - Receive Ack (t2)
    - Inactivity (ti)
- Connection Retry Limits
    - Maximum Retries
    - Delay After Failure

Connection Properties: Address

Remote Network Address

A 12-digit hexadecimal network address. Contact the remote system's administrator or the network administrator for the

correct value.

The default is **400000000000**.

The value is determined as follows:

- For connections to a 3174 controller, use the value in the configuration response 900 of the controller's customization program.
- For connections to a 3720, 3725, or 3745 front-end processor, use the value in the MACADDR= parameter in the NCP configuration.
- For connections to an IBM 9370 host, use the value in the VTAM PORT definition for the MACADDR= parameter.
- For connections to another Host Integration Server system, the network address can be found through the **net config server** command. When this command is typed at the command prompt of the other Host Integration Server system, the resulting display shows the network address of that system, in the line labeled **Server is Active On**.

Remote Service Access Point (SAP) Address

A two-digit hexadecimal number that is a multiple of 04, between 04 and EC. See your token ring or Ethernet manual for more information.

The default is **04**.

You can also contact the remote system's administrator for information about the following:

- For connections to a 3174 controller, use the value in the configuration response 940 of the controller's customization program.
- For connections to an IBM 9370 host, use the value in the VTAM PU definition for the SAPADDR= parameter.

Local Service Access Point (SAP) Address

A two-digit hexadecimal number that is a multiple of 04, between 04 and EC. See your token ring or Ethernet manual for more information.

The Default is **04**.

You can also contact the remote system's administrator for information about the following:

- For connections to a 3174 controller, use the value in the configuration response 940 of the controller's customization program.
- For connections to an IBM 9370 host, use the value in the VTAM PU definition for the SAPADDR= parameter.

Affiliate Application

If you selected Single Sign-On, choose an Affiliate Application from the list. The Enterprise Single Sign-On (SSO) Affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line of business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

Connection Properties: DLC 802.2

Max BTU Length

Specify the Maximum Basic Transmission Unit (BTU) Length.

The range is from 265 through 16393; the default is **265**.

With an IBM SDLC adapter, set the Max BTU Length to 521 or less.

This is the maximum number of bytes that can be transmitted in a single data-link information frame. A BTU is sometimes called an I-frame.

For downstream connections, specify a Max BTU Length less than or equal to the maximum value supported by software on the downstream system.

For host connections, specify a Max BTU Length less than or equal to the VTAM PU definition for the MAXDATA= parameter.

Receive ACK Threshold (frames)

Specify the maximum number of frames that the local system can receive from the remote system before sending a response. Set this to a value less than the **Unacknowledged Send Limit**, so that the local system acknowledges received transmissions more frequently than it expects responses to sent transmissions.

Unacknowledged Send Limit (frames)

Specify the maximum number of frames that the local system can send without receiving a response from the remote system. This parameter is sometimes called the Send Window Size.

Retry Limit

Specify the number of times that the local system should retransmit a frame if no response is received from the remote system.

The range is from 0 through 255; the default is **10**. A value of 0 means the system uses its internal default retry limit.

XID Retries

Specify the number of times that the local system should retransmit an XID (an identifying message) if no response is received from the remote system.

The range is from 0 through 30; the default is **3**.

Response (t1) Timeout

Select a value for the Response (t1) Timeout.

The amount of time that the local system should wait for the remote system to respond to a transmission before the local system tries again.

The values used for Default for the Response Timeout are 400 milliseconds for a local ring and 2 seconds for a remote ring.

If you do not select Default, but instead select a specific value, the timer always waits the selected amount of time, regardless of whether the remote system is on a local or remote ring.

Select a value greater than the total amount of time needed for the relaying of frames between the local system, the remote system, and the network.

Select **Default** to allow for two timeout values — one for a remote system on a local ring, one for a remote system on a remote ring. If you do not select **Default**, but instead select a specific value, the timer always waits the selected amount of time, regardless of whether the remote system is on a local or remote ring.

Receive ACK (t2) Timeout

Select a value for the Receive ACK (t2) Timeout.

The maximum amount of time that should be allowed before the local system sends an acknowledgment of a received transmission. An acknowledgment is sent at the end of the timeout, if some other process has not already triggered it.

Select a value less than the Response Timeout, so that the local system takes less time to acknowledge received transmissions than it takes to seek responses to sent transmissions.

Select **Default** to allow for two timeout values — one for a remote system on a local ring, one for a remote system on a remote ring. If you do not select **Default**, but instead select a specific value, the timer always waits the selected amount of time, regardless of whether the remote system is on a local or remote ring.

The values used for default for the Receive ACK Timeout are 80 milliseconds for a local ring and 800 milliseconds for a remote ring.

Inactivity (ti)

Select a value for the Inactivity (ti) Timeout.

The amount of time that the link can be inactive before the local system treats it as nonfunctioning and shuts it down.

Select **Default** to allow for two timeout values — one for a remote system on a local ring, one for a remote system on a remote ring. If you do not select **Default**, but instead select a specific value, the timer always waits the selected amount of

time, regardless of whether the remote system is on a local or remote ring.

The values used for default for the Inactivity Timeout are 5 seconds for a local ring and 25 seconds for a remote ring.

Connection Retry Limits

Supply activation limits for the connection.

Maximum Retries

Select the number of attempts for Host Integration Server to make when trying to establish the connection. After making this number of attempts, Host Integration Server makes an entry in the event log and stops trying. The range is from 1 through No Limit; the default is **No Limit**.

Delay After Failure

Select the length of time for Host Integration Server to wait between attempts to establish the connection. The range is from 5 seconds through 255 seconds; the default is **10** seconds.

Connection Properties: X.25

The following tabs are available on the X.25 Connection Properties sheet:

Connection Properties: Address

Remote X.25 Address

Select the type of virtual circuit used by the connection.

Switched Virtual Circuit Address

Enter the Switched Virtual Circuit Address of the host.

Permanent Virtual Circuit Alias

Enter the Permanent Virtual Circuit Alias of the host. This is a number that identifies the channel: 1 for the first channel, 2 for the second, and so on. The default is **1**.

Affiliate Application

If you selected Single Sign-On, choose an Affiliate Application from the list. The Enterprise Single Sign-On (SSO) Affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line of business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

Connection Properties: X.25

Max BTU Length

Specify the Maximum Basic Transmission Unit (BTU) Length.

The range is from 265 through 16393; the default is **265**.

With an IBM SDLC adapter, set the Max BTU Length to 521 or less.

This is the maximum number of bytes that can be transmitted in a single data-link information frame. A BTU is sometimes called an I-frame.

For downstream connections, specify a Max BTU Length less than or equal to the maximum value supported by software on the downstream system.

For host connections, specify a Max BTU Length less than or equal to the VTAM PU definition for the MAXDATA= parameter.

Facility Data

For an SVC channel, specify the codes for any Facility Data required by your network provider or by the administrator of the remote system. You can specify as many as 126 Hexadecimal characters (63 Hexadecimal bytes).

User Data

For an SVC channel, specify the codes for any User Data required by your network provider. Type an even number of hexadecimal characters of 32 characters or fewer.

Packet Size

For a PVC channel, select the maximum number of data bytes (not header bytes) to be sent in a frame. Obtain this value from your network provider.

Window Size

For a PVC channel, select the maximum number of frames that the local system can send without receiving a response from the remote system. Obtain this value from the administrator of the remote host or peer system.

Connection Retry Limits

Supply activation limits for the connection:

Maximum Retries

Select the number of attempts for Host Integration Server to make when trying to establish the connection. After making this number of attempts, Host Integration Server makes an entry in the event log and stops trying. The range is from 1 through No Limit; the default is **No Limit**.

Delay After Failure

Select the length of time for Host Integration Server to wait between attempts to establish the connection. The range is from 5 seconds through 255 seconds; the default is **10** seconds.

See Also

Reference

X.25 Connection Parameters

Connection Properties: Channel

The following tabs are available on the Channel Connection Properties sheet:

Connection Properties: Address

Channel Address

Type a two-digit hexadecimal number identifying the channel. The range is from 00 through FF; the default is **FF**.

Control Unit Image Number

Type a value for the control unit image number. The range is 0 through F; the default is **0**.

Affiliate Application

If you selected Single Sign-On, choose an Affiliate Application from the list. The Enterprise Single Sign-On (SSO) Affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line of business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

Connection Properties: Channel

Max BTU Length

Specify the Maximum Basic Transmission Unit (BTU) Length.

The range is from 265 through 16393; the default is **265**.

With an IBM SDLC adapter, set the Max BTU Length to 521 or less.

This is the maximum number of bytes that can be transmitted in a single data-link information frame. A BTU is sometimes called an I-frame.

For downstream connections, specify a Max BTU Length less than or equal to the maximum value supported by software on the downstream system.

For host connections, specify a Max BTU Length less than or equal to the VTAM PU definition for the MAXDATA= parameter.

Connection Retry Limits

Supply activation limits for the connection.

Maximum Retries

Select the number of attempts for Host Integration Server to make when trying to establish the connection. After making this number of attempts, Host Integration Server makes an entry in the event log and stops trying. The range is from 1 through No Limit; the default is **No Limit**.

Delay After Failure

Select the length of time for Host Integration Server to wait between attempts to establish the connection. The range is from 5 seconds through 255 seconds; the default is **10** seconds.

Local LU Properties: General

Local APPC LUs can be independent or dependent.

LU Alias

Enter the LU Alias.

Network Name

Type the Network Name. Obtain the correct name from the host or peer administrator if you will be connecting to a host system with VTAM,

The Network Name for an APPC LU that communicates with a host should match the NETID parameter in the VTAM Start command for the VTAM system.

If this server communicates with several different hosts over several connections, use the name of the subarea of the host with which the LU will communicate.

For independent LUs, the Network Name is required. For dependent LUs, the Network Name is recommended but not required, since it is used only by local software, such as the Windows event log software.

LU Name

Enter the LU Name. For independent APPC, the LU Name identifies the LU to other components on the SNA network, and therefore is required. For dependent APPC, the LU Name identifies the LU to local software, such as the Windows event log software, and is recommended but not required.

The LU Name and LU Alias for an APPC LU can be the same.

Comment

Optionally, enter a comment of no more than 25 characters.

Local LU Properties: Advanced

Member of Default Outgoing Local APPC LU Pool

If you want this LU to be available for use by a 5250 user or invoking TP not specifying a local LU, select this check box.

When a request comes from an invoking TP, and the request does not specify a local LU for the invoking TP to use, Host Integration Server first checks the user record of the user who started the TP, and tries to use the default local APPC LU assigned to that user or group. If this does not succeed, Host Integration Server tries to find a free LU in the pool of Default Outgoing Local APPC LUs. If this in turn does not succeed, the request is rejected.

Local LUs only support the number of sessions configured for the mode being used. The default for QPCSUPP is 64 sessions. If you need more than that number of sessions, then you need to configure multiple local LUs or increase the session limits in the mode definition for each mode used. To simplify user configuration, you can make all of these local APPC LUs part of the default pool by checking the **Member of Default Outgoing Local LU Pool.** This allows any user who does not specify a local APPC LU to get an available session from any local APPC LU in the default pool. This also enables load balancing among local APPC LUs. In addition, to ensure proper load balancing, do not specify a **Default Local LU Alias** for users or groups. However, if you want to have certain users or groups default to a certain local APPC LU, then you should specify that local APPC LU as the **Default Local LU Alias** in the user or group properties.

This default pool differs from, and should not be confused with, the 3270, LUA, and downstream LU pools.

Timeout for Starting Invokable TPs

If the **Invokable TP** is started manually by an operator, specify a value greater than 60 seconds to give the operator sufficient time.

Implicit Incoming Remote LU

To specify an Implicit incoming remote LU, choose an existing remote LU name from the list.

LU 6.2 Type

Select **Independent** or **Dependent**.

LU Number

For dependent LUs, enter the LU Number.

For independent LUs, this field is unavailable.

Connection

For **Dependent LU**s, choose the connection from the drop-down list.

For **Independent LU**s, this field is unavailable.

LU 6.2 Resync Service

Computer

Type the **IP Address** or the **Name** of the client computer. The client computer specifies a system that is dedicated in that the LU routes incoming connections to that client computer.

Check the **Enable** box if you have a very specialized transaction program (TP) that requires Resync Service. Resync Service or SyncPoint support is used by some database management systems (DBMS) for commit and rollback procedures. If you enable this service, the Local LU alias must be unique.

If you have multiple local and remote LUs using two-phase commit, you may want to explicitly partner the LUs. This will prevent the Resync Service from attempting to bind invalid LU pairs.

To explicitly partner LUs

1. In **SNA Manager**, click the **APPC Modes** folder.
2. Right-click **RSYNPRTN**, and then click **Properties**.
3. Select the **Partner** tab.
4. Click **Add**, and then follow the directions in the dialog box.

If you would like the Resync Service to use a different mode name, you can specify a new name with the following registry key:

HKLM\Software\Microsoft\Host Integration Server\UN2

REG_SZ: modename

Remote LU Properties: General

Connection

From the drop-down list, choose the connection that will be used to access this remote LU.

LU Alias

Enter an LU Alias

Network Name

Enter a Network Name. Obtain the correct name from the host or peer administrator. If you will be connecting to a host system with VTAM, the Network Name for an APPC LU that communicates with a host should match the NETID parameter in the VTAM Start command for the VTAM system.

LU Name

Enter the LU Name.

A name identifying an LU. The name can be from one through eight characters long and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The name cannot be the same as any other LU name or pool name (except for APPC LU names) on the server.

For communication with an AS/400, make the remote LU name the same as the name of the AS/400.

Uninterpreted LU Name

If this LU will be paired with a dependent local APPC LU, type the Uninterpreted LU Name.

Comment

Optionally, enter a comment of not more than 25 characters.

Remote LU Properties: Options

Supports Parallel Sessions

If this LU will be used with an independent local LU, and parallel sessions will be used, select this check box. If this LU will be used with a dependent local APPC LU, do not select this box.

If a remote LU supports parallel sessions, it can only be used with a mode that has a value greater than 1 for the parallel session limit.

Implicit Incoming Mode

To designate a mode as the Implicit Incoming Mode for this LU, select a mode from the list.

If this remote LU will be used as an Implicit Incoming Mode be sure to select a mode from the list.

No Session-Level Security

Select this option to turn off Session-Level Security.

Security Key in Hex

For a security key in hexadecimal, select this option, then type a 16-digit hexadecimal number.

Security Key in Characters

For a security key in characters, select this option, then type an eight-character string. The string can include uppercase and lowercase alphanumeric characters, and the special characters $, @, #, and the period (.).

Enable SyncPoint

If you enable **SyncPoint**, the Local LU alias must be unique.

Server Configuration Properties

Name

Displays the current server name. Click **Change** to change the server name.

Properties

The properties display the current configuration. These properties are display only. To change any of the following configuration items, click **Change**.

Subdomain

The **Subdomain** displays the current SNA subdomain.

Server Role

The server can be either a primary, backup, or member. The current role of the server is displayed.

Network Transports

Network Transports shows the transport currently in use.

Support Active Directory Clients

If selected, this server supports client computers using Active Directory and displays the current Active Directory Domain Name/Organization Unit.

Server Configuration

Subdomain Name
Displays the SNA subdomain that this server is a member of. To change the subdomain name, enter a new subdomain name and click **Save**.

Active Directory Client Support
If selected, this server is part of the Active Directory Organizational Unit (OU) as shown in the **Active Directory OU Name:** field. To change the OU, enter a new OU name and click **Save**.

Role
Select how this server will be configured, either as a **Primary Configuration Server** or a **Backup Configuration Server**.

Network Transports
Select all network transport protocols that this server will use on the network.

Services
The **Services** box shows the status of the **MngAgent** and **SnaBase** services. If these services are running, you must restart them before changes to the subdomain, server role, or network transports will take effect.

Restart
Click **Restart** to restart services required for changes to the subdomain, server role, or network transports to take effect. **Restart** is active only if you change the configuration of another server. If you want to change the configuration of the server on which you are currently working, **Restart** is unavailable. For changes to take effect on your server, close SNA Manager, open **Control Panel**, open **Services**, and then **Stop** and **StartSNA MngAgent** and **SnaBase**.

Save
Click **Save** to save your server configuration changes.

Important
Changing server configuration parameters can have serious consequences. Restarting the server will cause all connections a nd sessions to be disconnected. Notify all users prior to restarting.

Workstation Properties: General

Workstation ID

The **workstation ID** is usually the workstation name.

Workstation ID Type

Click the type of workstation ID:

Name is the workstation name.

Address is the IP address. An IP Address specifies one server for a network address (for example, IP, IPX, or Vines).

IP Subnet is the IP Subnet address. An IP Subnet address specifies several servers with the same network address. Subnetting enables host administrators to distribute the hostids for a given netid to several subnetworks. Without subnetting, an IP address is interpreted in two parts: netid + hostid (network + node). With subnetting, an IP address is interpreted in three parts: netid + subnetid + hostid (network + subnetwork + node). Subnetting is a mechanism for using some bits in the hostid as a subnetid.

IP Mask

If you select **IP Subnet**, you must specify the **IP Mask** as well. The IP Mask refers to the netid + subnetid. You must know the IP Mask. For example, an IP Mask might be: "255.255.248.0."

Comment

Optionally, enter a comment of no more than 25 characters.

Allow access to both workstation and user resources

Select this option to allow users logged on to this computer to see the LUs assigned both to the workstation and to that user.

Allow access to workstation resources only

Selecting this option provides a higher degree of security by restricting resources available on this computer to the LUs assigned to this workstation. Users will not be able to access LUs assigned to them unless the LU is also assigned to the workstation. This feature will be useful for secure workstations, such as those used to print checks.

Allow access to Dynamically Created Remote APPC LUs

Select this check box to allow users access to remote APPC LU as they are created.

TN5250 Properties

Name

Displays the name of the server running the TN5250 service. This field cannot be edited here.

Comment

Optionally, type a comment of up to 25 characters.

Use Default

Select **Use Default** to use the port number associated with telnet (as set in \\DRIVERS\ETC\SERVICES).

Use

You can override the default value for a given server by selecting **Use** and typing another port number. You can also override this port number for a given session. TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client computer to connect to multiple host computers.

Host Integration Server, VTAM, and NCP Parameters for X.25 Connections

The following table shows how Host Integration Server parameters for X.25 connections correspond to VTAM or NCP parameters. Asterisks (*) indicate required parameters.

Properties Tab	**SNA Parameter**	**VTAM or NCP Parameter**
System Identification	**Local Node ID**,* first three digits (block number)	**IDBLK=** parameter in the PU definition
System Identification	**Local Node ID**,* last five digits (node number)	**IDNUM=** parameter in the PU definition
System Identification	**Network Name (Remote Node)** Note: Network Name for local and remote nodes is generally the same	**NETID=** parameter in the VTAM Start command for the remote SSCP (VTAM system)
System Identification	**Control Point Name (Remote Node)**	**SSCPNAME=** parameter in the VTAM Start command for the remote SSCP (VTAM system)
Address	**Virtual Circuit Type or Remote X.25 Address**	**DIALNO=** parameter in the PORT definition
X.25	**Max BTU Length**	**MAXDATA=** parameter in the PU definition (set Max BTU Length less than or equal to MAXDATA)

- Required parameter in Host Integration Server.

Domain Properties

The following tabs are available for the SNA Subdomain Properties sheet:

Domain Properties: NetView

NetView sends alerts back and forth between a host system and the Host Integration Server and associated client computers.

Send NetView Management Data to this Connection

Select a connection to which NetView data should be sent. This can be any host connection.

Domain Properties: Display Verb

Default Connection for Display

Select a connection for use by Display Verbs. When the Display Verb is used but does not specify a connection, Host Integration Server uses the connection you specify in **Display Verb Properties**. If you do not specify a default Display connection, Host Integration Server randomly selects a connection for the verb to use.

Domain Properties: Server Broadcasts

Servers communicate with each other using server broadcasts.

Mean Time between Server Broadcasts

The interval at which server broadcasts are repeated. Broadcasts of state changes (such as the activation of a server) do not wait for this interval; instead, broadcasts are repeated at this interval to ensure reception (since broadcasts by definition are not guaranteed to be received).

Specifying a larger value causes less demand on the network (since broadcasts occur less often). However, a smaller value compensates better for loss of broadcast messages.

The range is from 45 through 65535 seconds; the default is **60** seconds.

Network Transport for server Broadcasts

Choose only the protocols you need, because sending unneeded broadcasts can significantly decrease network efficiency.

Domain Properties: Security

An administrator can adjust the subdomain security settings, including user permissions, configuration file ownership, and subdomain security events to audit.

LU Types

Select the **LU Types** (3270, LUA, or APPC) to which the security settings will apply. If security for an LU type is off, anyone who knows the LU name can use it. If security is on, only users to whom the LU is assigned can use it. The default setting is on for 3270, and off for LUA and APPC.

Configuration File

To view or change user permissions on the configuration file, click **Permissions**.

To view or change audited events, click **Auditing**.

To view or change configuration file ownership, click **Take Ownership**.

Domain Properties: Client Backup Configuration

Disable Sending Client Backup Information

The server sends no information to the client computer.

Backup Domain

The server updates the client computer with the backup SNA subdomain name specified in the box.

Backup Sponsor Servers

The server updates the client computer with all of the SNA Sponsor Server names (shown in the box). The maximum is 15.

Domain Properties: Error / Audit Logging

You can use information from the Windows 2000 event logs as you test a configuration or to diagnose a problem.

Centralized Event Log Server

Select the name of the server on which Windows 2000 event logs for this server installation should be stored.

Route All Popup Messages to

Select the name of the server to which pop-up error messages should be routed.

Default Audit Log Level

Select a level

Detailed problem analysis: To record all events that can be recorded, select this option.

General information messages: To record general activity but not all events, select this option.

Significant system events: To record only major events, select this option.

Audit logging disabled: To turn off audit logging, select this option.

Domain Properties: Response Time Monitor (RTM)

Response Time Monitor tracks the time it takes a host to respond to 3270 session requests. RTM is supported only by certain emulators. You should configure RTM only if the emulators you are using support RTM.

RTM Data Sent at

Select one or both of the following:

Counter Overflow:

To cause RTM data to be sent to the host when the number of host responses in a given time period overflows the size of the available counter, select this box.

End of Session:

To cause RTM data to be sent to the host at the end of each LU-to-LU session, select this box.

RTM Timers Run Until

Select the point at which RTM will register that a host has responded; this is when RTM stops the timers. (The timers are started when the local system sends data.)

First Data Reaches Screen:

To stop timing when data reaches the local screen, select this option.

Host Unlocks Keyboard:

To stop timing when the host unlocks the local keyboard, select this option.

Host Lets User Send:

To stop timing when the host lets the local computer send more data, select this option.

RTM Thresholds

Specify the cutoff times, in tenths of a second, at which RTM saves its count of host responses, and then restarts the count. For example, you could specify 5, 10, 20, and 50, to save the count of host responses during the intervals from 0.0 to 0.5 seconds, from 0.5 to 1.0 seconds, from 1.0 to 2.0 seconds, and from 2.0 to 5.0 seconds.

AS400 Definition Properties: General

AS/400 Remote LU Alias

Click the drop-down list box arrow and select an **AS/400 Remote LU Alias**, which contains addressing information for the AS/400.

Local LU Alias

Click the drop-down list box arrow and select a **Local LU Alias**. The **Local LU Alias** maps to the LU the client computer will use.

Mode

You must select the **QPCSUPP mode**.

AS/400 User Name

Enter your **AS/400 User Name**, which is required information.

AS/400 Password

Enter your **AS/400 Password**, which is required information.

Confirm Password

Enter your password again to confirm.

Comment

Optionally, enter a comment of not more than 25 characters long.

Active Users

This folder displays the users that are currently actively using this server.

LUA LU Properties: General

After assigning or viewing an LUA LU, in the **LUA LU Properties** box, supply information according to the following list.

LU Number (for LUs on 802.2, SDLC, or X.25)
Type the LU Number.

LU Name
Type the LU Name.

Connection
The connection for this LU is shown. The connection cannot be changed in this dialog box.

Pool
If the LU has already been assigned to a pool, the pool name appears here.

Comment
Optionally, type a comment of 25 characters or fewer.

CPI-C Symbolic Name: General

Name

Type the Symbolic Destination Name.

Comment

Optionally, enter a comment of not more than 25 characters.

Conversation Security

Select a Conversation Security option:

- **None** turns off conversation security.
- **Same** requires the same user ID and password as that of the Remote Partner LU.
- **Program** specifies the user ID and password, select Program, and then click **User ID**.

Mode Name

Select a mode name from the drop-down list.

CPI-C Symbolic Name: Partner Information

Partner TP Name

- **Application TP:** For an Application TP, select this option, and type the Application TP Name.
- **SNA service TP** (in hex)**:** For an SNA service TP select this option, and type the SNA service TP Number.Partner LU Name
- **Alias:** To identify the **Partner LU by** Alias, select this option, and type the alias.
- **Fully Qualified:** To identify the partner LU by a Fully Qualified name, select this option, and type the name.

To accept the settings, click **OK**; to exit the dialog box without accepting the settings, click **Cancel**.

CPI-C Symbolic Name: Security Settings

Specify a User ID and Password

User ID
The **User ID** can contain from 1 through 10 characters.

Password
The **Password** can contain from 1 through 10 characters.

Verify Password
Type the **password** again.

Host Integration Server 2009 Folder

Expand the Servers folder to show the Host Integration Server assigned to this SNA Domain.

SNA Service Folder

This folder contains the SNA service installed on this Host Integration Server computer. Each Host Integration Server computer can have a maximum of four SNA services installed and active at one time.

Select **SNA Service**, then right-click to perform various tasks, including:

- Save the current configuration.
- Stop the service.
- Pause the service.
- Start the service.
- Create a new service.
- Display additional service properties.

Link Services Folder

This folder contains the link services that are configured on this Host Integration Server.

To create a new link service, select the **Link Services** folder, right-click, point to **New** and then select **Link Service**.

Connections Folder

This folder contains the connections configured on this Host Integration Server.

To create a new connection, select the **Connections** folder, right-click, point to **New** and then select the type of connection you want to make.

Local APPC LUs Folder

This folder contains the Local APPC LUs configured for this server.

Remote APPC LUs Folder

This folder contains the Remote APPC LUs configured for this server.

Microsoft SNA Manager

Host Integration Server Microsoft Management Console application is also called the SNA Manager.

AS/400 Definitions

This folder displays the AS/400 definitions that are currently being used on this server.

Active TN5250 Sessions

This folder displays the active TN5250 sessions that are using this server.

AS/400 Definition Properties

The following tabs are available on the AS/400 Definition Properties:

AS/400 Definition Properties: General

AS/400 Remote LU Alias

Click the drop-down list box arrow and select an **AS/400 Remote LU Alias**, which contains addressing information for the AS/400.

Local LU Alias

Click the drop-down list box arrow and select a **Local LU Alias**. The **Local LU Alias** maps to the LU the client computer will use.

Mode

You must select the **QPCSUPP mode**.

System 36, AS/36

Select if connection to a System 36 or AS/36 computer system.

AS/400 User Name

Enter your **AS/400 User Name**, which is required information.

AS/400 Password

Enter your **AS/400 Password**, which is required information.

Confirm Password

Enter your password again to confirm.

Comment

Optionally, enter a comment of not more than 25 characters long.

AS/400 Definition Properties: Terminal Types

Terminal Names

Terminal names are displayed. You can remove selection of individual terminal names that do not apply to your network configuration by clearing the checkbox opposite the name

Port

Use Default

Select **Use Default** to use the port number configured with the service (on the Service Properties page).

Use

You can override the default value for a given session by selecting **Use** and typing another port number.

Note
TN services listen on multiple ports simultaneously. You can set a default number for the TN service and override this numbe r on a per session basis, allowing a single client computer to connect to multiple host computers.

Note
Configuration changes are apparent only to users who establish a connection after the configuration changes are saved. User s who were connected at the time the configuration changes were made will not be affected.

AS/400 Definition Properties: IP Address List

You can associate this LUA to a specific IP address or server name.

Add Address

Click to specify the **IP Address** and **Subnet Mask** of the client workstation to which you are granting access.

IP Address

Click an entry in the **IP Address** column to edit the IP address for a client workstation.

Subnet Mask

Click an entry in the **Subnet Mask** column to edit the subnet mask for a client workstation.

Add Name

Click to add a workstation name to the **IP Address** list box. The workstation name identifies that a client workstation whose address resolves to that name can connect to the service.

Remove

Click to remove an entry that is currently highlighted in the **IP Address** list box.

Clear All

Click to remove all entries in the **IP Address** list box. The service will then be configured to allow access from any client workstation.

Note

The IP address must be changed from the default value of 0.0.0.0 if you want to assign a specific IP address. If you try to add the default IP address with the default subnet mask value, you will receive a message indicating an Invalid IP Address/Subnet Mask.

Note

The IP address of an incoming connection is compared to each of the available LUs or pools in turn. Each resource has an associated list of IP address patterns and subnet masks. The IP address from the incoming connection is masked (by bit and by the resource's subnet mask), then compared to the resource's IP address pattern masked by the resource's subnet mask. If the result is equal, the connection is allocated to this resource.

APPC Modes Folder

This folder contains the APPC Modes configured for this server.

CPI-C Symbolic Names Folder

This folder contains the CPIC Symbolic Names configured for this server.

Visual Studio Help

Microsoft Visual Studio provides a graphical user interface for creating, viewing, and editing the objects used by the host-initiated processing (HIP) and Windows-initiated processing (WIP) environments in Transaction Integrator (TI). TI objects are created and edited within a TI Project, one of several project types supported by the Visual Studio design environment.

The user interface for a TI Project includes a tree view in the Solution Explorer window, wizards, dialog boxes, and properties displayed in the Properties window. The tree view presents a two-pane design tool. The left pane displays a hierarchical representation of the components of a type library or .NET assembly. The right pane displays a list of details about the properties associated with the contents of the library or assembly selected in the left pane. You can customize the details view to display other types of information, for example, a COBOL, RPG, or IDL source code. You can display Help topics about the individual user interface controls by highlighting a control or node and pressing the F1 key.

The TI wizards in a project have **Help** buttons on each wizard page. Clicking **Help** on the page takes you to the relevant Help topic in this section. You can display Help topics about the individual user interface controls by clicking **Help** on the Visual Studio wizard page or dialog box or by highlighting a control or node and pressing the F1 key. All of the TI Project Help topics are combined in this section for easy reference and review.

In This Section

COM Library Nodes

When you view a COM type library (.tlb for client and .tim for server) from within a TI Project, the Visual Studio design environment displays the objects in the library in a tree view in the left pane and a details view in the right pane. Double-click the node to expand the node and right-click the node to display a shortcut menu.

The topics in this section describe the functionality of each menu and each command in the navigation tree.

In This Section

- Library Name Node
- Interface Name Node (COM)
- Method Name Node (COM)
- Parameter Name Node (COM)
- Recordsets Node
- Recordset Name Node
- Recordset Column Name Node
- User-Defined Types Node
- User-Defined Type Name Node
- User-Defined Type Member Name Node

Library Name Node

Use the ***library name*** node to support Microsoft Visual Basic 6.0, transactions, and select the remote environment class.

Double-click the ***library name*** node to expand it. The right pane displays the following information about the node's children:

- **Name**.The object name.

Right-click the ***library name*** node to display the following nine options:

- **Import**. Displays a menu with two options:
    - **Host Definition**. Starts the Import COBOL Wizard or RPG Import Wizard to help you import COBOL or RPG to the interface definition.
    - **Library**. Starts the **Import Library** open file dialog box to help you to import an existing type library or assembly to the interface definition.
- **Export Host Definition**. Starts the **Export Host Definition** file save dialog box to help you generate a COBOL or RPG copy book equivalent to the current library.
- **Lock**. Marks the library as read-only. The library is automatically marked as locked if it is registered in a COM+ application or IIS virtual directory.
- **Rename**. Renames the library.
- **Properties**. Displays the **Properties** window.

See Also

Reference

Library Properties
Interface Name Node (COM)
Recordsets Node
User-Defined Types Node

Interface Name Node (COM)

Use the ***interface name*** node to set the name of the interface in a component.

Double-click the ***interface name*** node to expand it. The right pane displays the following information about the interface methods:

- **Method Name**. The method name.
- **Return Type**. The method return type.
- **Host Data Type**. The COBOL or RPG equivalent of the method's return type.
- **Array Size(s)**. If the return type is an array, this column will contain the number of dimensions and their sizes. For example, a single dimension of size 10 will be displayed as (10); a 3-dimensions array with sizes 2, 4, and 6 will be displayed as (2, 4, 6).
- **Rows**. The number of recordset rows if the return type of the method is a recordset.
- **Link-to-Program Name**. The name of an executable running under the host environment that will be linked to by this method and passed a COMMAREA. It is valid only for link models.
- **TP Name**. The name of the transaction program (TP) used by the method to locate a program to be executed. In the case of link models, it identifies the mirror transaction identifier which parses the distributed program link (DPL) header and identifies the link-to-program name.
- **Meta Data**. Indicates whether the host is sent no metadata, only the name of the method, or all metadata.

Right-click the ***interface name*** node to view the following seven options:

- **Add Method**. Adds a new method .
- **Paste**. Inserts a method into the current interface definition.
- **Rename**. Renames the interface.
- **Properties**. Displays the **Properties** window.

See Also

Reference

Interface Properties
Method Name Node (COM)
Library Name Node

Method Name Node (COM)

Use the ***method name*** node to define the conditions for interacting with the host environment, including the value for the initial buffers, whether to include all information.

Double-click the ***method name*** node to expand it. The right pane displays the following information about the method parameters:

- **Parameter Name**. The name of the parameter.
- **Type**. The data type of the parameter.
- **Direction**. The direction of the parameter: in only, in and out, or out only.
- **Host Data Type**. The COBOL or RPG data type that best describes the parameter type.
- **Array Size(s)**. The size and number of the dimensions of the parameter, if it is an array.
- **Rows**. The number of recordset rows if the parameter type is a recordset.

Right-click the ***method name*** node to view the following seven options:

- **Add Parameter**. Adds a new In\Out integer parameter to the method.
- **Cut**. Copies the selected method to the Clipboard and deletes it.
- **Copy**. Copies the selected method to the Clipboard.
- **Paste**. Inserts a parameter from the Clipboard into the current method definition.
- **Delete**. Deletes the method.
- **Rename**. Renames the method.
- **Properties**. Displays the **Properties** window.

See Also

Reference

Method Properties
Parameter Name Node (COM)
Interface Name Node (COM)

Parameter Name Node (COM)

Use the ***parameter name*** node to define the information passed as a parameter and what precision and scale is to be used.

Double-click the ***parameter name*** node to expand it. The right pane does not display any additional information.

Right-click the ***parameter name*** node to view the following eight options:

- **Move Up**. Moves a parameter up in the list.
- **Move Down**. Moves a parameter down in the list.
- **Cut**. Copies the selected parameter to the Clipboard and deletes it.
- **Copy**. Copies the selected parameter to the Clipboard.
- **Paste**. The paste command is disabled because parameters do not have any child elements.
- **Delete**. Deletes the selected parameter.
- **Rename**. Renames the selected parameter.
- **Properties**. Displays the **Properties** window.

See Also

Reference

Parameter Properties

Recordsets Node

Use the **recordsets** node to define the table passed as a parameter or return value.

Double-click the **recordsets** node to expand it. The right pane displays the following information about the defined recordsets:

- **Name**. The name of each recordset.

Right-click the **recordsets** node to view the following seven options:

- **Add Recordset**. Adds a new recordset definition.
- **Paste**. Inserts a recordset column from the Clipboard into the current folder.
- **Properties**. Displays the **Properties** window.

See Also

Reference

Recordset Properties
Recordset Name Node
Library Name Node

Recordset Name Node

Use the ***recordset name*** node to define the name of the recordset.

Double-click the ***recordset name*** node to expand it. The right pane displays the following information about the recordset columns:

- **Recordset Column Name**. The name of the recordset column.
- **Type**. The data type of the recordset column.
- **Host Data Type**. The COBOL or RPG data type that best describes the recordset column type.

Right-click the ***recordset name*** node to view the following seven options:

- **Add Recordset Column**. Adds a new integer column to the selected recordset.
- **Cut**. Copies the contents of the recordset to the Clipboard and deletes it.
- **Copy**. Copies the contents of the recordset to the Clipboard.
- **Paste**. Inserts the column from the Clipboard into the current recordset definition.
- **Delete**. Deletes the current recordset.
- **Rename**. Renames the recordset.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Recordset Properties
Recordset Column Name Node

Recordset Column Name Node

Use the ***recordset column name*** node to define the recordset column.

Double-click the ***recordset column name*** node to expand it. No information is displayed in the right pane.

Right-click the ***recordset column name*** node to view the following eight options:

- **Move Up**. Moves a column up in the list.
- **Move Down**. Moves a column down in the list.
- **Cut**. Copies the selected recordset column to the Clipboard and deletes it.
- **Copy**. Copies the selected recordset column to the Clipboard.
- **Paste**. The paste command is disabled because recordset columns do not have child elements.
- **Delete**. Deletes the recordset column.
- **Rename**. Renames the recordset column.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Recordset Column Properties

User-Defined Types Node

Use the **User-Defined Types** (UDT) node to define a structure that contains the pieces of data.

Double-click the **User-Defined Types** node to expand it. The right pane displays the following information about the defined UDTs:

- **Name**. The name of each user-defined type.

Right-click the **User-Defined Types** node to view the following seven options:

- **Add User-Defined Type**. Adds a new user-defined type.
- **Paste**. Inserts the UDT from the Clipboard into the current library definition.
- **Properties**. Displays the **Properties** window.

See Also

Reference

User-Defined Type Properties
User-Defined Type Name Node
Library Name Node

User-Defined Type Name Node

Use the ***user-defined type name*** node to define the type of data the user-defined type (UDT) is carrying.

Double-click the ***user-defined type name*** node to expand it. The right pane displays the following information about the UDT members:

- **User-Defined Type Member Name**. The name of the user-defined member.
- **Type**. The data type of the user-defined member.
- **Host Data Type**. The COBOL or RPG data type that best describes the user-defined member type.
- **Array Size(s)**. The size and number of the dimensions of the user-defined member, if it is an array.
- **Rows**. The number of recordset rows, if the user-defined member type is a recordset.

Right-click the ***user-defined type name*** node to view the following seven options:

- **Add User-Defined Type Member**. Adds a new integer member.
- **Cut**. Copies the contents of the user-defined type to the Clipboard and deletes it.
- **Copy**. Copies the contents of the user-defined type to the Clipboard.
- **Paste**. Inserts the member from the Clipboard into the user-defined type definition.
- **Delete**. Deletes the current user-defined type.
- **Rename**. Renames the user-defined type.
- **Properties**. Displays the **Properties** window.

See Also
Reference
User-Defined Type Properties
User-Defined Type Member Name Node
User-Defined Types Node

User-Defined Type Member Name Node

Use the ***user-defined type member name*** node to view information about the UDT member.

Double-click the ***user-defined type member name*** node to expand it. No information is displayed in the right pane.

Right-click the ***user-defined type member name*** node to view the following eight options:

- **Move Up**. Moves a user-defined type (UDT) member up in the list.
- **Move Down**. Moves a user-defined type member down in the list.
- **Cut**. Copies the selected user-defined type member to the Clipboard and deletes it.
- **Copy**. Copies the selected user-defined type member to the Clipboard.
- **Paste**. The paste menu item is disabled because user-defined type members do not have child elements.
- **Delete**. Deletes the user-defined type member.
- **Rename**. Renames the user-defined type member.
- **Properties**. Displays the **Properties** window.

See Also
Reference
User-Defined Type Member Properties
User-Defined Type Name Node

.NET Framework Library Nodes

When you view a .NET library (.dll for client and .tim for server) from within a TI Project, the Visual Studio design environment displays the objects in the library in a tree view in the left pane and list view in the right pane. Double-click the node to expand the node and right-click the node to display a shortcut menu.

The topics in this section describe the functionality of each menu and each command in the navigation tree.

In This Section

- Interface Name Node (.NET)
- Method Name Node (.NET)
- Parameter Name Node (.NET)
- Data Tables Node
- Data Table Name Node
- Data Table Member Name Node
- Structures Node
- Structure Name Node
- Structure Member Name Node

Interface Name Node (.NET)

Use the ***interface name*** node to view the list of all the methods in a component.

Double-click the ***interface name*** node to expand it. The right pane displays the following information about the methods:

- **Method Name**. The method name.
- **Return Type**. The method return type.
- **Host Data Type**. The COBOL or RPG equivalent of the method's return type.
- **Array Sizes**. If the return type is an array, this column will contain the number of dimensions and their sizes. For example, a single dimension of size 10 will be displayed as (10); a 3-dimensions array with sizes 2, 4, and 6 will be displayed as (2, 4, 6).
- **Rows**. The number of rows, if the return type of the array is a data table.
- **Link-to-Program Name**. The name of an executable running under the host environment that will be linked to by this method and passed a COMMAREA. It is valid only for link models.
- **TP Name**. The transaction program (TP) name used by the method to locate a program to be executed. In the case of link models, it identifies the mirror transaction identifier which parses the distributed program link (DPL) header and identifies the link-to-program name.
- **Meta Data**. Indicates whether the host is sent no metadata, only the name of the method, or all metadata.

Right-click the ***interface name*** node to view the following seven options:

- **Add Method**. Adds a new method with no return type.
- **Paste**. Inserts the method from the Clipboard into the current interface definition.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Interface Properties
Method Properties
Library Name Node
Method Name Node (.NET)

Method Name Node (.NET)

Use the ***method name*** node to view the list of all the methods parameters.

Double-click the ***method name*** node to expand it. The right pane displays the following information about the parameters:

- **Parameter Name**. The name of the parameter.
- **Type**. The data type of the parameter.
- **Direction**. The direction of the parameter: in only, in and out, or out only.
- **Host Data Type**. The COBOL or RPG data type that best describes the parameter type.
- **Array Size(s)**. The size and number of dimensions of the parameter, if it is an array.
- **Rows**. The number of rows, if the parameter type is a data table.

Right-click the ***method name*** node to display the following seven options:

- **Add Parameter**. Adds a new integer parameter to the method.
- **Cut**. Copies the contents of the method to the Clipboard and marks it as deleted.
- **Copy**. Copies the contents of the method to the Clipboard.
- **Paste**. Inserts the parameter from the Clipboard into the current method definition.
- **Delete**. Deletes the current method.
- **Rename**. Renames the current method.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Parameter Name Node (.NET)

Parameter Name Node (.NET)

Use the ***parameter name*** node to change the name of the parameter or view its properties.

Double-click the ***parameter name*** node to expand it. No information is displayed in the right pane.

Right-click the ***parameter name*** node to display the following eight options:

- **Move Up**. Moves a parameter up in the list.
- **Move Down**. Moves a parameter down in the list.
- **Cut**. Copies the contents of the parameter to the Clipboard and deletes it.
- **Copy**. Copies the contents of the parameter to the Clipboard.
- **Delete**. Deletes the current parameter.
- **Rename**. Renames the current parameter.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Parameter Properties
Method Name Node (.NET)

Data Tables Node

Use the **Data Tables** node to view a list of all the **Data Tables** in the assembly.

Double-click the **Data Tables** node to expand it. The right pane displays the following information about the data tables:

- **Name**. Name of the individual data tables.

Right-click the **Data Tables** node to view the following seven options:

- **Add Data Table**. Adds a new data table to the assembly.
- **Paste**. Inserts the data table from the Clipboard into the current data table definition.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Data Table Name Node
Library Name Node

Data Table Name Node

Use the ***data table name*** node to view a list of the data table column names and their properties.

Double-click the ***data table name*** node to expand it. The right pane displays the following information about the data table columns:

- **Data Table Column Name**. The name of the table column.
- **Type**. The data type of the data table.
- **Host Data Type**. The COBOL or RPG data type that best describes the data table type.

Right-click the ***data table name*** node to display the following seven options:

- **Add Data Table Column**. Adds a new column to the data table.
- **Cut**. Copies the contents of the data table to the Clipboard and deletes it.
- **Copy**. Copies the contents of the data table to the Clipboard.
- **Paste**. Inserts the column from the Clipboard into the current data table definition.
- **Delete**. Deletes the current data table.
- **Rename**. Renames the current data table.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Data Table Member Name Node
Data Tables Node

Data Table Member Name Node

Use the ***data table member name*** node to change the name of the data table member or view its properties.

Double-click the ***data table member name*** node to expand it. No information is displayed in the right pane.

Right-click the ***data table member name*** node to display the following six options:

- **Cut**. Copies the contents of the data table to the Clipboard and marks it as deleted.
- **Copy**. Copies the contents of the data table to the Clipboard.
- **Paste**. Inserts the contents of the Clipboard into the current data table column definition.
- **Delete**. Deletes the current data table .
- **Rename**. Renames the current data table .
- **Properties**. Displays the **Properties** window.

See Also
Reference
Data Tables Node

Structures Node

Use the **Structures** node to view a list of all the **Structures** in an assembly.

Double-click the **Structures** node to expand it. The right pane displays the following information about the structures:

- **Name**. Name of the individual structures.

Right-click the **Structures** node to display the following seven options:

- **Add Struct**. Adds a new structure to the assembly.
- **Paste**. Inserts the structure from the Clipboard into the current assembly definition.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Structure Name Node
Library Name Node

Structure Name Node

Use the ***structure name*** node to view a list of structure member names and their properties.

Double-click the ***structure name*** node to expand it. The right pane displays the following information about the structure members:

- **Structure Member Name**. The name of the structure member.
- **Type**. The data type of the data table.
- **Host Data Type**. The COBOL or RPG data type that best describes the data table type.
- **Array Size(s)**. The size and number of dimensions of the data table, if it is an array.
- **Rows**. The number of rows, if the data table type is a data table.

Right-click the ***structure name*** node to view the following seven options:

- **Add Structure Member**. Adds a new structure member.
- **Cut**. Copies the contents of the structure to the Clipboard and deletes it.
- **Copy**. Copies the contents of the structure to the Clipboard.
- **Paste**. Inserts the member from the Clipboard into the current structure definition.
- **Delete**. Deletes the current structure.
- **Rename**. Renames the current structure.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Structure Member Name Node
Structures Node

Structure Member Name Node

Use the ***structure member name*** node to change the name of the structure member or view its properties.

Double-click the ***structure member name*** node to expand it. No information is displayed in the right pane.

Right-click the ***structure member name*** node to display the following six options:

- **Cut**. Copies the contents of the structure member to the Clipboard and deletes it.
- **Copy**. Copies the contents of the structure member to the Clipboard.
- **Delete**. Deletes the current structure member.
- **Rename**. Renames the current structure member.
- **Properties**. Displays the **Properties** window.

See Also
Reference
Structures Node

Wizards and Dialog Boxes (TI Project)

In This Section

New COM Server Library Wizard

The **New COM Server Library Wizard** collects information about the type library information and about the host environment (HE). The wizard generates a Transaction Integrator metadata (TIM) file and adds it to the current TI Project displayed in the Solution Explorer.

In This Section

- Welcome to the New COM Server Library Wizard Page
- Library Wizard Page (COM Server Wizard)
- Host Environment Wizard Page (COM Server Wizard)
- Completing the New COM Server Library Wizard Page

Welcome to the New COM Server Library Wizard Page

Use the **Welcome to the New COM Server Library Wizard** page to view the definition of a COM server library and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Library Wizard Page (COM Server Wizard)

Library Wizard Page (COM Server Wizard)

Use the **Library** wizard page to identify the COM server library you are creating.

Use this	To do this
Interface name	Type the name for the interface in the type library. The name of the interface, when combined with the major version number and the library name (ProgID), cannot exceed 39 Unicode characters.
Version	Type the major version information for the type library. The major version number, when combined with name of the interface and the library name (ProgID), cannot exceed 39 Unicode characters.
Component ProgID	Displays the program identifier to be associated with the CoClass in the type library. The format is *Library.Interface.MajorVersion*. The ProgID cannot exceed 39 Unicode characters.
Description	Type the description to be associated with the interface .The description can be a maximum of 256 Unicode characters.
Support Visual Basic 6.0 decimal data type	Select this option if you want the component to allow application programs written with Visual Basic 6.0 and reference decimal data types.

See Also

Reference

Library Properties

Host Environment Wizard Page (COM Server Wizard)

Host Environment Wizard Page (COM Server Wizard)

Use the **Host Environment** (HE) wizard page to select the HE that defines the network and hardware characteristics of the non-Windows software platform initiating requests to the Windows platform. The HE consists of the host environment name, host identification, network transport type, data conversion information, default method resolution criteria, and security credential mapping.

Use this	To do this
Host environment	Select the host environment (HE). The available HEs are: • **Transaction Integrator - ConvertPrim for OS390** (default) • **Transaction Integrator - ConvertPrim for AS400**

See Also
Reference
Completing the New COM Server Library Wizard Page
Library Properties

Completing the New COM Server Library Wizard Page

Use the **Completing the New COM Server Library Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also

Other Resources

New COM Server Library Wizard

New COM Client Library Wizard

The **New COM Client Library Wizard** collects information about the type library information and about the remote environment (RE). The wizard generates an annotated Windows-initiated processing (WIP) COM type library (.tlb) and adds the library to the current TI Project displayed in the Solution Explorer.

In This Section

Welcome to the New COM Client Library Wizard Page

Use the **Welcome to the New COM Client Library Wizard** page to view the definition of a COM client library and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Library Wizard Page (COM Client Wizard)

Library Wizard Page (COM Client Wizard)

Use the **Library** wizard page to identify the COM client library you are creating.

Use this	To do this
Interface name	Type the name for the interface in the type library. The name of the interface, when combined with the major version number and the library name (ProgID), cannot exceed 39 Unicode characters.
Version	Type the major version information for the type library. The major version number, when combined with the name of the interface and the library name (ProgID), cannot exceed 39 Unicode characters.
Component ProgID	View the program identifier to be associated with the CoClass in the type library. The format is *Library.Interface.MajorVersion*. The ProgID cannot exceed 39 Unicode characters.
Description	Type the description to be associated with the interface. The description can be a maximum of 256 Unicode characters.
Support Visual Basic 6.0 decimal data type	Select this option if you want the component to allow application programs written with Visual Basic 6.0 and reference decimal data types.

See Also

Reference

Remote Environment Wizard Page 1 (COM Client Wizard)
Remote Environment Wizard Page 2 (for OS400) (COM Client Wizard)
Remote Environment Wizard Page 2 (for LU 6.2 Link) (COM Client Wizard)
Completing the New COM Client Library Wizard Page
Library Properties

Remote Environment Wizard Page 1 (COM Client Wizard)

Use the **Remote Environment** wizard page to define the remote environment (RE). A remote environment identifies an application execution environment running on a remote host system. The remote environment can have a data communication protocol and a programming model associated with it.

Use this	To do this
Vendor	Select the name of the vendor supplying the remote environment.
Protocol	Select the data communication protocol to be used to connect to the remote environment. The available communication protocols are: • **TCP/IP** (default) • **LU 6.2**
Target environment	Select the application environment running on the remote host system. The available environments are: • **CICS** (default) • **IMS** • **OS400** (TCP/IP only)
Programming model	Select the programming model associated with the remote environment. The available models for the TCP/IP CICS target environment are: • **TRM User Data** (default) • **TRM Link** • **ELM User Data** • **ELM Link** The available models for the LU 6.2 CICS target environment are: • **CICS User Data** • **CICS Link** • The available models for the TCP/IP IMS target environment are: • **IMS Connect** • **Implicit** • **Explicit** • The only available model for the LU 6.2 IMS target environment is **IMS User Data**. • The only available model for the OS400 target environment is **Distributed Program Call**.

Allow 32 K in/out	Select this option if you want TI to treat the input DFHCOMMAREA independently from the output DFHCOMMAREA. TI typically combines the input DFHCOMMAREA and the output DFHCOMMAREA area. The combined areas cannot exceed 32 KB of data. When this option is selected, TI treats the input DFHCOMMAREA independently from the output DFHCOMMAREA. Each input and output area uses up to 32 KB of data.

See Also

Reference

Library Properties

Completing the New COM Client Library Wizard Page

Concepts

TCP Transaction Request Message Link

TCP Enhanced Listener Message Link

TCP Transaction Request Message User Data

TCP Enhanced Listener Message User Data

Remote Environment Wizard Page 2 (for OS400) (COM Client Wizard)

Use the **Remote Environment** wizard page to define the default date and time separators for the remote environment (RE).

Use this	To do this
Program library	Type the name of the program library.
Date separator	Select the character used to separate the parts of the date. The available separators are: **(none)** (default) **/** **-** **.** **,** **&**
Time separator	Select the character used to separate the parts of time. The available separators are: **(none)** (default) **:** **.** **,** **&**

See Also

Reference

Library Properties

Completing the New COM Client Library Wizard Page

Concepts

OS/400 Distributed Program Calls

Remote Environment Wizard Page 2 (for LU 6.2 Link) (COM Client Wizard)

Use the **Remote Environment** wizard page to define the default values for the remote environment (RE).

Use this	To do this
Transaction ID	Type the transaction program (TP) name or the link mirror transaction ID of the remote environment. The name or ID can be a maximum of 4 alphabetic characters. The format of the name must conform to the IBM SNA Transaction Program Names convention. The default is **CSMI**.
Source TP name	Type the name of the transaction program used by the method to locate a program to be executed. In the case of link models, the name identifies the mirror transaction identifier, which parses the distributed program link (DPL) header and identifies the link-to-program name. The name can be a maximum of 250 Unicode characters. The default is **MSTX**.

See Also

Reference

Library Properties

Completing the New COM Client Library Wizard Page

Concepts

TCP Transaction Request Message Link

TCP Enhanced Listener Message Link

TCP Transaction Request Message User Data

TCP Enhanced Listener Message User Data

Completing the New COM Client Library Wizard Page

Use the **Completing the New COM Client Library Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also

Other Resources

New COM Client Library Wizard

New .NET Client Library Wizard

The **New .NET Client Library Wizard** collects information about the assembly information and about the remote environment (RE). The Wizard generates an annotated .NET assembly and adds the assembly to the current TI Project displayed in the Solution Explorer.

In This Section

Welcome to the New .NET Client Library Wizard Page

Use the **Welcome to the New .NET Client Library Wizard** page to view the definition of a .NET client library and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Library Wizard Page (.NET Client Wizard)

Library Wizard Page (.NET Client Wizard)

Use the **Library** wizard page to identify the .NET client library you are creating.

Use this	To do this
Interface name	Type the full name for the class in the .NET assembly. The name of the interface, when combined with the major version number and the namespace name, cannot exceed 39 Unicode characters.
Version	Type the version information for the .NET assembly in the form *major.minor*. The major version number, when combined with the name of the interface and the namespace name, cannot exceed 39 Unicode characters.
Component class name	View the component class name to be associated with the managed class in the .NET assembly. The format is *Namespace.Interface.MajorVersion*. The namespace name, when combined with the major version number and the name of the interface, cannot exceed 39 Unicode characters.
Type Restrictions	Select the type appopriate to your library. As you continue to develop your project, appropriate restrictions will be applied based on this selection. This step will help prevent you from creating a library that contains types that are not supported by the applications that will be using them (for example, ASMX or BizTalk Server). Selecting **None** will result in no restrictions. Selecting **Web Service** will allow access to data sets, but restrict access to data tables and multi-dimensional arrays. Selecting **BizTalk Adapter for Host Applications** will allow access to arrays only, and restrict access to data tables, data sets, and multi-dimensional arrays.
Description	Type the description to be associated with the interface. The description can be a maximum of 256 Unicode characters.

See Also

Reference

Library Properties

Remote Environment Wizard Page 1 (.NET Client Wizard)

Remote Environment Wizard Page 1 (.NET Client Wizard)

Use the **Remote Environment** wizard page to define the remote environment (RE). An RE identifies an application execution environment running on a remote host system. The RE can have a data communication protocol and a programming model associated with it.

Use this	To do this
Vendor	Select the name of the vendor supplying the remote environment.
Protocol	Select the data communication protocol to be used to connect to the remote environment. The available communication protocols are: • **TCP/IP** (default) • **LU 6.2**
Target environment	Select the application environment running on the remote host system. The available environments are: • **CICS** (default) • **IMS** • **OS400** (TCP/IP only)
Programming model	Select the programming model associated with the remote environment. The available models for the TCP/IP CICS target environment are: • **TRM User Data** (default) • **TRM Link** • **ELM User Data** • **ELM Link** The available models for the LU 6.2 CICS target environment are: • **CICS User Data** • **CICS Link** • The available models for the TCP/IP IMS target environment are: • **IMS Connect** • **Implicit** • **Explicit** • The only available model for the LU 6.2 IMS target environment is **IMS User Data**. The only available model for the OS400 target environment is **Distributed Program Call**.

Allow 32 K in/out	Select this option if you want TI to treat the input DFHCOMMAREA independently from the output DFHCOMMAREA. TI typically combines the input DFHCOMMAREA and the output DFHCOMMAREA area. The combined areas cannot exceed 32 KB of data. When this option is selected, TI treats the input DFHCOMMAREA independently from the output DFHCOMMAREA. Each input and output area uses up to 32 KB of data.

See Also

Reference

Library Properties

Remote Environment Wizard Page 2 (for OS400) (.NET Client Wizard)

Remote Environment Wizard Page 2 (for LU 6.2 Link) (.NET Client Wizard)

Completing the New .NET Client Library Wizard Page

Concepts

TCP Transaction Request Message Link

TCP Enhanced Listener Message Link

TCP Transaction Request Message User Data

TCP Enhanced Listener Message User Data

Remote Environment Wizard Page 2 (for OS400) (.NET Client Wizard)

Use the **Remote Environment** wizard page to define the default values for the remote environment (RE).

Use this	To do this
Program library	Type the name of the program library.
Date separator	Select the character used to separate the parts of the date. The available separators are: **(none)** (default) **/** **-** **.** **,** **&**
Time separator	Select the character used to separate the parts of time. The available separators are: **(none)** (default) **:** **.** **,** **&**

See Also

Reference

Library Properties

Completing the New .NET Client Library Wizard Page

Remote Environment Wizard Page 2 (for LU 6.2 Link) (.NET Client Wizard)

Use the **Remote Environment** wizard page to define the default values for the remote environment (RE).

Use this	To do this
Transaction ID	Type the transaction program (TP) name or the link mirror transaction ID of the remote environment. The name or ID can be a maximum of 4 alphabetic characters. The format of the name must conform to the IBM SNA Transaction Program Names convention. The default is **CSMI**.
Source TP name	Type the name of the transaction program used by the method to locate a program to be executed. In the case of link models, the name identifies the mirror transaction identifier, which parses the distributed program link (DPL) header and identifies the link-to-program name. The name can be a maximum of 4 Unicode characters. The default is **MSTX**.

See Also

Reference

Library Properties

Completing the New .NET Client Library Wizard Page

Completing the New .NET Client Library Wizard Page

Use the **Completing the New .NET Client Library Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also
Other Resources
New .NET Client Library Wizard

New .NET Server Library Wizard

The **New .NET Server Library Wizard** collects information about the type library information and about the host environment (HE). The wizard generates a Transaction Integrator metadata (TIM) file and adds it to the current TI Project displayed in the Solution Explorer.

In This Section

- Welcome to the New .NET Server Library Wizard Page
- Library Wizard Page (.NET Server Wizard)
- Host Environment Wizard Page (.NET Server Wizard)
- Completing the New .NET Server Library Wizard Page

Welcome to the New .NET Server Library Wizard Page

Use the **Welcome to the New .NET Server Library Wizard** page to view the definition of a .NET server library and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Library Wizard Page (.NET Server Wizard)

Library Wizard Page (.NET Server Wizard)

Use the **Library** wizard page to identify the .NET server library you are creating.

Use this	To do this
Interface name	Type the full name for the class in the .NET assembly. The name of the interface, when combined with the major version number and the namespace name, cannot exceed 39 Unicode characters.
Version	Type the version information for the .NET assembly in the form *major.minor*. The major version number, when combined with name of the interface and the namespace name, cannot exceed 39 Unicode characters
Component class name	View the component class name to be associated with the managed class in the .NET assembly. The format is *Namespace.Interface.MajorVersion*. The namespace name, when combined with the major version number and the name of the interface, cannot exceed 39 Unicode characters
Description	Type the description to be associated with the interface. The description can be a maximum of 256 Unicode characters.

See Also

Reference

Library Properties

Host Environment Wizard Page (.NET Server Wizard)

Host Environment Wizard Page (.NET Server Wizard)

Use the **Host Environment** (HE) wizard page to select the HE that defines the network and hardware characteristics of the non-Windows software platform initiating requests to the Windows platform. The HE consists of the host environment name, host identification, network transport type, data conversion information, default method resolution criteria, and security credential mapping.

Use this	To do this
Host environment	Select the host environment (HE). The available HEs are: • **Transaction Integrator - ConvertPrim for OS390** (default) • **Transaction Integrator - ConvertPrim for AS400**

See Also

Reference

Completing the New .NET Server Library Wizard Page

Library Properties

Completing the New .NET Server Library Wizard Page

Use the **Completing the New .NET Server Library Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also

Other Resources

New .NET Server Library Wizard

Import COBOL Wizard

The **Import COBOL Wizard** allows you to select a COBOL copy book and use it for designing the component in the designer. This wizard is rather complex, as there are multiple options and paths to follow.

In This Section

Welcome to the Import COBOL Wizard Page

Use the **Welcome to the Import COBOL Wizard** page to view the procedure for importing a COBOL copy book and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Import COBOL Source File Wizard Page

Import COBOL Source File Wizard Page

Use the **Import COBOL Source File** wizard page to identify the COBOL source file.

Use this	To do this
COBOL copy book	Select the full path to the COBOL copy book to be imported.
COBOL source	View the COBOL source code contained in the file identified in **COBOL copy book**.

See Also

Reference

Item Options Wizard Page

Item Options Wizard Page

Use the **Item Options** wizard page to specify the type of component to create. Select and identify the type of component to be created.

Use this	To do this
Type of item	View the group of component types: • **Method** • **Recordset** • **User-defined type**
Select or type the name of the item to import	View the identifying information for the new component.
Name	Select or type the name of the component. The name can be a maximum of 250 Unicode characters.
Use Link programming model	Type the name of an executable program that, running under the host environment (HE), will be linked to by this method and passed a COMMAREA. This option is available only for Link models. The name can be a maximum of 8 Unicode characters. This option is displayed only for host-initiated processing (HIP).
Exclude LL, ZZ, and TRANCODE fields	Select to exclude the LL, ZZ, and TRANCODE fields from the transaction if the host environment is IMS. This option is available only for HIP server components.
Link-to-Program	The name of an executable running under the host environment that will be linked to by this method and passed a COMMAREA. The name can be a maximum of 8 Unicode characters. This option is available only for Windows-initiated processing (WIP) Link server components.
Source TP Name	The name of the mirror transaction identifier, which parses the distributed program link (DPL) header and identifies the link-to-program name. The name can be a maximum of 4 Unicode characters. This option is available only for WIP Link server components.

See Also

Reference

DFHCOMMAREA Wizard Page

DFHCOMMAREA Wizard Page

Use the **DFHCOMMAREA** wizard page to select the COBOL group that represents the DFHCOMMAREA parameters.

Use this	To do this
COBOL group	Select the group or parameter to include.

See Also

Reference

DFHCOMMAREA Direction Wizard Page

DFHCOMMAREA Direction Wizard Page

Use the **DFHCOMMAREA Direction** wizard page to view the default direction for the DFHCOMMAREA and to change the direction of a group item. To change the direction, click the icon to the left of the item, and then select the appropriate direction.

Use this	To do this
DFHCOMMAREA	Select the item to change.

See Also

Reference

Input Area Wizard Page

Input Area Wizard Page

Use the **Input Area** wizard page to select the input parameters for the new method in a non-link programming model.

Use this	To do this
Input area	Select or clear input parameters.

See Also

Reference

Output Area Wizard Page

Output Area Wizard Page

Use the **Output Area** wizard page to select the output parameters for the new method in a non-link programming model.

Use this	To do this
Output area	Select or clear output parameters.

See Also

Reference

Return Value Wizard Page

Return Value Wizard Page

Use the **Return Value** wizard page to set the return value for the new non-link method.

Use this	To do this
Return value	Select or clear return value.

See Also

Reference

Recordsets (COBOL) & UDT Wizard Page

Recordsets (COBOL) & UDT Wizard Page

Use the **Recordsets & UDT** wizard page to specify whether any of the parameters are represented by recordsets or UDTs. To change a group item, click the icon to the left of the group, and then select the definition.

Use this	To do this
COBOL Group	Select whether the group is defined as a recordset or UDT type. Select **Unset** to clear the definition.

See Also

Reference

LL Field Wizard Page

LL Field Wizard Page

Use the **LL Field** wizard page to select the LL fields to exclude from the transaction if the host environment (HE) is IMS.

Use this	To do this
LL field	Select or clear LL fields.

See Also

Reference

ZZ Field Wizard Page

ZZ Field Wizard Page

Use the **ZZ Field** wizard page to select the ZZ fields to exclude from the transaction if the host environment (HE) is IMS.

Use this	To do this
ZZ field	Select or clear ZZ fields.

See Also

Reference

TRANCODE Field Wizard Page

TRANCODE Field Wizard Page

Use the **TRANCODE Field** wizard page to select the TRANCODE fields to exclude from the transaction if the host environment (HE) is IMS.

Use this	To do this
TRANCODE field	Select or clear TRANCODE fields.

See Also

Reference

Recordset Columns Wizard Page

Recordset Columns Wizard Page

Use the **Recordset Columns** wizard page to select the COBOL structure that represents the recordset columns.

Use this	To do this
COBOL group	Select the COBOL group to include, or clear the check box to exclude the group.

See Also

Reference

User-Defined Type Members Wizard Page

User-Defined Type Members Wizard Page

Use the **User-Defined Type Members** wizard page to select the COBOL structure that represents the user-defined type (UDT) members.

Use this	To do this
COBOL group	Select the COBOL group to include, or clear the check box to exclude the group.

See Also

Reference

Completing the Import COBOL Wizard Page

Completing the Import COBOL Wizard Page

Use the **Completing the Import COBOL Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also
Other Resources
Import COBOL Wizard

Import RPG Wizard

The **Import RPG Wizard** allows you to import the definitions from Report Program Generator (RPG) source code written for use by distributed program call (DPC) applications running on AS/400 computers.

In This Section

- Welcome to the Import RPG Wizard Page
- Import RPG Source File Wizard Page
- Select Item Wizard Page
- PLIST Direction Wizard Page
- Recordsets and User-Defined Types (RPG) Wizard Page
- Completing the RPG Import Wizard Page

Welcome to the Import RPG Wizard Page

Use the **Welcome to the Import RPG Wizard** page to view the procedure for importing Report Program Generator (RPG) source code and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Import RPG Source File Wizard Page

Import RPG Source File Wizard Page

Use the **Import RPG Source File** wizard page to identify the file that contains the Report Program Generator (RPG) source statements. After being identified, the RPG source statements are displayed in the edit control. Select the RPG file to import.

Use this	To do this
RPG source file	Select or type the full path to the RPG file to import.
RPG source	View the RPG source code contained in the file identified in **RPG source file**.

See Also

Reference

Select Item Wizard Page

Select Item Wizard Page

Use the **Select Item** wizard page to create a new method or rewrite an existing method in your interface. Enter the identifying information for the method.

Use this	To do this
Method name	Select or type the name of the COM or .NET method. The name can be a maximum of 250 Unicode characters. If you are creating a new method, the name of the new method cannot be the same as the name of an existing method.
Program name	Select or type the name of the AS/400 Report Program Generator (RPG) program. The name can be a maximum of 10 Unicode characters.
Library name	Select or type the name of the library on the AS/400 computer from which the named RPG program will be retrieved for execution. The name can be a maximum of 10 Unicode characters.

See Also

Reference

PLIST Direction Wizard Page

PLIST Direction Wizard Page

Use the **PLIST Direction** wizard page to select the direction of the PLIST parameters. By default, all parameters are set to In/Out. To change the direction of a group item, click the icon to the left of the name, and then select the appropriate direction.

Use this	To do this
PLIST	Click the *Report Program Generator* (RPG) parameter to change, and then click the new direction.

See Also
Reference
Recordsets and User-Defined Types (RPG) Wizard Page
Other Resources
Import RPG Wizard

Recordsets and User-Defined Types (RPG) Wizard Page

Use the **Select Recordsets and User-Defined Types** wizard page to specify whether a parameter is represented by recordsets or user-defined types (UDTs). To change a group item, click the icon to the left of the group, and then select the definition.

Use this	To do this
Group	Select whether the group is defined as a recordset or user-defined type. Select **Unset** to clear the definition.

See Also

Reference

Completing the RPG Import Wizard Page

Completing the RPG Import Wizard Page

Use the **Completing the RPG Import Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also

Other Resources

Import RPG Wizard

Properties (TI Project)

In This Section

Library Properties

Use the **Library** properties page to set design and remote environment (RE) properties on the component library or .NET assembly.

Design Properties

Use this	To do this
Component Type	View the type of library component. Possible values are: • **COM Type Library** (default) • **.NET Assembly**
Description	Type a description of the component. The description can be a maximum of 250 Unicode characters.
Help Context ID	Type the Help context ID associated with this method. It is used to connect to the Help topic for this method, and returned when an exception occurs during invocation of this method.
Initiation Type	View the location of the transaction program (TP) making the request. Possible values are: • **Windows-initiated** (default) • **Host-initiated**
Name	Type the name of the component. The name can be a maximum of 250 Unicode characters. The name must be different from any other component name in the same project. The default is **Library(n)**.
Support Visual Basic 6.0 Decimal Type	Select this option to specify whether the component allows application programs written with Visual Basic 6.0 that reference decimal data types. The possible values are: • **True** • **False** (default)
Visual Basic 6.0 Compatible	Select this option to require that the Visual Basic server use the **Implements** keyword within the Visual Basic server script. The server script then creates the Visual Basic interface to the Transaction Integrator metadata (TIM) type library. The possible values are: • **True**. Use the **Implements** keyword. • **False** (default) Note This property is the same as the property **Visual Basic Implements Compatible** in Host Integration Server.
Version	Type the version information for the type library or the .NET assembly, expressed in the form of *major.minor*.

Remote Environment Properties

Use this	To do this

Allow 32K In/Out	Select this option if you want TI to treat the input DFHCOMMAREA independently from the output DFHCOMMAREA. TI typically combines the input DFHCOMMAREA and the output DFHCOMMAREA area. The combined areas cannot exceed 32 KB of data. When this option is selected, TI treats the input DFHCOMMAREA independently from the output DFHCOMMAREA. Each input and output area uses up to 32 KB of data. Changing this option affects the currently selected method. Possible values are: • **True** • **False** (default)
Remote Environment Class	Select the programming model associated with the remote environment. The available models for the TCP/IP CICS target environment are: • **TRM User Data** (default) • **TRM Link** • **ELM User Data** • **ELM Link** The available models for the LU 6.2 CICS target environment are: • **CICS User Data** • **CICS Link** • The available models for the TCP/IP IMS target environment are: • **IMS Connect** • **Implicit** • **Explicit** • The only available model for the LU 6.2 IMS target environment is **IMS User Data**. The only available model for the OS400 target environment is **Distributed Program Call**.

Transaction Support	The possible values are: • **Required**. The new TI component is used in applications that execute within COM+ transactions. The component's methods will be called by applications used with mainframe transaction programs that support Sync Level 2 requests. If a transaction is in progress, the application will enlist in the transaction. Otherwise, the application will start a new transaction. • **Required New**. The new TI component is used in applications that execute within COM+ transactions. The component's methods will be called by applications used with mainframe transaction programs that support Sync Level 2 requests. The application will always start a new transaction, regardless of whether an existing transaction is already in progress. • **Supported**. The new TI component is used in applications that might or might not execute within COM+ transactions. The component's methods will be called by applications used with mainframe transaction programs that support both Sync Level 0 (non-transactional) and Sync Level 2 (transactional) requests. In the case of a Sync Level 2 transaction, the calling application will enlist in the transaction if the transaction is already in progress. Otherwise, the application will start a new transaction. • **Not Supported**. The new TI component is used in applications that do not execute within COM+ transactions. The component's methods will be called by applications used with mainframe transaction programs that support only Sync Level 0 requests. IMS transaction programs prior to IMS 6.0 support only Sync Level 0. CICS and IMS 6.0 transaction programs support either Sync Level 0 or Sync Level 2 requests.

⚠Warning

The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

Library Name Node

Interface Properties

Use the **Interface** properties page to set design properties on the interface.

Design Properties

Use this	To do this
Name	Type the name of the library component. The name can be a maximum of 250 Unicode characters.

⚠Caution
The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties p rogrammatically might cause the component to function incorrectly.

See Also

Reference

Interface Name Node (COM)

Interface Name Node (.NET)

Method Properties

Use the **Method** properties page to set array, COBOL, design, host definition, and recordset properties on the method.

Array Properties

Use this	To do this
Array Dimensions	Select the array dimensions for the return value. The default is **(none)**.
Is Array	Select whether the return value is an array. The possible values are: • **True** • **False** (default)
Occurs Depending On	Select to indicate that a numeric data item preceding the table (recordset or array in Automation) indicates the actual number of rows or elements being sent or received. Equivalent to variable-length tables in COBOL. Use the drop-down list to select the numeric data item that specifies this value. For CICS Link, the recordset or array and the associated length specifier must be in/out. The data in the buffer that follows a variable length table immediately follows the last data item in the table regardless of the maximum size specified for the table. Arrays with multiple dimensions can only be used for the outermost loop (COBOL) or rightmost dimension (Microsoft® Visual C++® or Visual Basic). The default is **(none)**.

COBOL Properties

Use this	To do this
HostDataType	Select the host data type.
ErrorHandling	Select the return value error handling. The possible values are: • **Truncate**. Select this option to set TI to truncate the value when an error occurs. (default) • **Round**. Select this option to set TI to round the value when an error occurs. • **Error**. Select this option to set TI to return an error when an error occurs.

Filler	Type the return value filler. The default is **0**.
FromHost	Type the number of bytes of FILLER that follows this data item in the buffers that are received from the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side. The default is **0**.
ToHost	Type the number of bytes of FILLER that follows this data item in buffers that are sent to the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side. The default is **0**.
Scale	Type the return value scale.
SignAttribute	Select the return value sign attribute. The possible values are: • **Trailing**. For signed DISPLAY data type, indicates that the sign is trailing (default). This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Trailing Separate**. For signed DISPLAY data type, indicates that the sign is separate. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Leading**. For signed DISPLAY data type, indicates that the sign is leading. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Leading Separate**. For signed DISPLAY data type, indicates that the sign is separate.
Size	Type the return value size.
SOSI	Select this option to specifywhether a double-byte character set data is expected to begin with a shift-out (SO) and end with a shift-in (SI) character. The possible values are: • **True**. The SO and SI characters are removed from the data when it is received from the host application, and the SO and SI characters are added to the data when it is sent to the host application. In the length of the PIC G, it is not necessary to include the two bytes for the SO and SI characters because the TI run-time environment applies them. • **False** (default)

String Delimiting	Select the return value string delimiting. The possible values are: • **Space-padded**. Tells the TI run-time environment that the mainframe representation of the string is delimited by padding the string definition with space characters. For example, if the mainframe's COBOL definition is PIC X(10) but only three characters are in the string, the mainframe expects seven trailing spaces. Therefore, selecting this option tells the TI run-time environment to convert strings being sent to the mainframe to change the string's NULL termination character to the appropriate number of trailing spaces before sending it to the mainframe. For example, if the string is defined on the mainframe as PIC X(10), TI will send a string of *ABC* followed by seven trailing spaces. Selecting this option also tells the TI run-time environment to convert the output string being returned from the mainframe to the TI Automation server by converting the string's trailing spaces to a single null termination character. For more information, see Padding Mainframe Character Strings with Spaces. (default) • **Null-terminated**. Tells the TI run-time environment that the mainframe representation of the string is delimited by a null character (EBCDIC 0x00). Selecting this option tells the TI run-time environment to add a single null character to the end of a string if there is room for the byte before sending a string to the mainframe, and it tells the TI run-time environment to stop at the first null character encountered when receiving a string from the mainframe. Therefore, by selecting this option, you are telling TI to retain trailing spaces in output strings coming from the mainframe because TI will not convert the trailing spaces to a single NULL terminator. For more information, see Padding Mainframe Character Strings with Spaces.

Design Properties

Use this	To do this
Allow 32K In/Out	Select this option if you want TI to treat the input DFHCOMMAREA independently from the output DFHCOMMAREA. TI typically combines the input DFHCOMMAREA and the output DFHCOMMAREA area. The combined areas cannot exceed 32 KB of data. When this option is selected, TI treats the input DFHCOMMAREA independently from the output DFHCOMMAREA. Each input and output area uses up to 32 KB of data. Changing this option affects the currently selected method. Possible values are: • **True** • **False** (default) Note You can use this property as an accessory to "Use Link Programming Model" in the Windows-initiated processing (WIP) CICS programming model and in any host-initiated processing (HIP) programming models. Note This property is available only if the **Is Link** property is set to **True**.
Description	Type a description of the method. The description can be a maximum of 250 Unicode characters.
Help Context ID	Type the Help context ID associated with this method. The ID is used to connect to Help for this method, and returned when an exception occurs during invocation of this method. The default is **0**.

Include Context Parameter	Select whether the client object method automatically includes context. The possible values are: • **True**. Visual Basic .NET automatically includes the context as an additional parameter in the argument. If you do not include **COMTIContext** parameter in your method call along with your other parameters, Visual Basic .NET returns the error message **An unhandled exception of type 'System.MissingMemberException' occurred in microsoft.visualbasic.dll** and informs you that the method cannot be called with the number of parameters you have written. If you receive this message, verify that the **Include Context Parameter** is included as a parameter within the list of parameters on the method. • **False**. Visual Basic .NET does not automatically include the context as an additional parameter in the argument. If you set this property to **False** and include **COMTIContext** parameter in your method call along with your other parameters, Visual Basic .NET returns the error message **An unhandled exception of type 'System.MissingMemberException' occurred in microsoft.visualbasic.dll** and informs you that the method cannot be called with the number of parameters you have written. If you receive this message, remove the **COMTIContext** parameter from the method parameter list. The default is **True**.
Initial Buffer Value	Type the initial buffer value. The default is null.
Is Link	Select whether the host object method uses the Link programming model. The possible values for Windows-initiated processing (WIP) are: • **True**. Use the link model. The link programming model can be used only with CICS link protocols. • **False**. Do not use the link model. The default is **False**. The possible values for host-initiated processing (HIP) are: • **Yes**. Use the link model. The link programming model can be used with all protocols. • **No**. Do not use the link model. • **Link using 32K In/Out**. Use the link model and set the From Host and To Host properties. The default is **No**.
Meta Data	Select how metadata is handled. The possible values are: • **(none)**. By default no special data is sent to or received from the host application. Select this option button if you only want to send and receive the data for the method. • **Include Method Information**. The name of this method to be sent to the host along with parameter data. The method name is sent as the first 32 bytes in the buffer. This option is useful if multiple method calls go to the same transaction and you want to differentiate the data from the different calls. • **Include All Information**. All metadata available to be sent and received with your method data. For details of the format of metadata, see description for "Optional Metadata".

Name	Type the name of the method. The name can be a maximum of 250 Unicode characters. The name must be different from any other method name in the same project. The default is **null**.
Position Return Value After	Type the Automation method return value that follows the selected data item when it is received from the host. This option does not affect the Automation side. Use this option when the data item that you want to specify as the Automation return value is not the first data item field in the data declaration that describes the data received from the host.
Preliminary Filler	View the number of bytes of FILLER received from or sent to the host.
From Host	Type the number of bytes of FILLER that follows this data item in the buffers that are received from the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
To Host	Type the number of bytes of FILLER that follows this data item in the buffers that are sent to the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
Return Type	Select the return value type. The possible values are: • **Void** • **Boolean** • **Byte** • **Date** • **Currency** • **Decimal** • **Integer** • **Long** • **Double** • **Single** • **String** • **User-defined type** • **Recordset** • **(none)** (default)

Variable Sized Final Field	• Select this option when the last data item is a string to indicate that the size of the string varies. This option is also used to define a datatable or recordset as being either bounded or as including all rows as defined as **Maximum Occurrence** set on the parameter.
From Host	• **True** • **False** (default)
To Host	• **True** • **False** (default)

Host Definition Properties

Use this	To do this
Link-to-Program Name	Type the link-to-program name (CICS LINK/DPL).
Mirror Transaction ID	Type the mirror TRANID that this method uses, if you want to override the mirror TRANID for the remote environment (RE) that this component is associated with. Leaving this box blank causes the mirror TRANID in the remote environment description to be used. The TRANID can be up to four characters in length. Acceptable characters are A-Z a-z 0-9 $ @ # . / _ % & ? ! : | = , ; < >. Transaction names beginning with C are reserved for CICS and should not be used. The % and & characters may cause problems with Resource Access Control Facility (RACF) if transaction security is active.
TP Name	Type a source transaction program (TP) name when the CICS application program must access a DB2 database. The TP name is referenced in a CICS Resource Control Table (RCT) entry, which associates CICS transactions with DB2 plans. Specifies the name of the host transaction program (IMS or CICS) or the link-to program name (CICS LINK/DPL).

Recordset Properties

Use this	To do this
Include Actual Size	The host program will not include or expect any information that indicates the actual number of rows (recordsets) or elements (arrays) being sent or received. The possible values are: • **True** • **False** (default) This property is read-only and will always be set to **False** unless it was set to **True** at the time the type library was created with the first version of COM Transaction Integrator.

Maximum Occurrence	Maximum row occurrence. Indicates the maximum number of rows to be sent to or received from the host. Equivalent to the OCCURS n TIMES keyword on a COBOL group item. The default is **1**.
Occurs Depending On	Equivalent to variable-length tables in COBOL. Indicates that a numeric data item preceding the table (recordset or array in Automation) indicates the actual number of rows or elements being sent or received. Use the drop-down list to select which numeric data item specifies this value. For CICS Link, the recordset or array and the associated length specifier must be in/out. The data in the buffer that follows a variable length table immediately follows the last data item in the table regardless of the maximum size specified for the table. For arrays with multiple dimensions, it can only be used for the outermost loop (COBOL) or rightmost dimension (Visual C++ or Visual Basic). Return value recordset occurs depending on. The default is **(none)**.
Unbounded	Indicates the recordset is unbounded. Indicates that any number of rows can be sent to or received from the host. You would select this option when the rows being sent or received are from a database and the maximum number of rows is not known. The possible values are: • **True**. When the last data item is a string, this means that the size of the string varies. • **False**. When the last data item is an array, this means that the number of elements in the array varies. When the last data item is a recordset, this means that the number of rows in the recordset varies. (default)

⚠Caution

The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

Method Name Node (COM)

Method Name Node (.NET)

Concepts

Using Custom TRMs and ELMs with COMTIContext

Parameter Properties

Use the **Parameter** properties page to set the array, host, COBOL design, and recordset properties on a parameter.

Array Properties

Use this	To do this
Array Dimensions	Use this control to indicate how many dimensions (Visual C++ or Visual Basic) or nested OCCURS clauses (COBOL) that the array or table contains.
Is Array	Select this option to indicate whether the parameter is an array. The possible values are: • **True**. Parameter is an array. • **False**. The item is a simple data type including RDA recordset objects. (default).
Occurs Depending On	Select this option to indicate that a numeric data item preceding the table (recordset or array in Automation) indicates the actual number of rows or elements being sent or received. Use the drop-down list to select which numeric data item specifies this value. For CICS Link, the recordset or array and the associated length specifier must be in/out. The data in the buffer that follows a variable length table immediately follows the last data item in the table regardless of the maximum size specified for the table. For arrays with multiple dimensions, it can only be used for the outermost loop (COBOL) or rightmost dimension (Visual C++ or Visual Basic). The RPG language, unlike COBOL, does not directly support **Occurs Depending On**. TI provides a feature that replicates the Occurs Depending On action for the RPG language. A TI Project supports a single level of dimension applied as an Occurs Depending On associated with a array of records in RPG. An index parameter must be defined prior to defining the parameter associated with a datatable or structure for the parameter to show as an **Occurs Depending On** selectable choice.

Host Properties

Use this	To do this
Host Data Type	Specifies the parameter host data type.

ErrorHandling	Parameter error handling. The possible values are: • **Truncate**. If selected and an error occurs, TI will truncate the value. (default) • **Round**. If selected and an error occurs, TI will round the value. • **Error**. If selected and an error occurs, TI will return an error.
Filler	Indicates the number of bytes of FILLER that follow this data item in the buffers that are sent to or received from the host. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
FromHost	Indicates the number of bytes of FILLER that follows this data item in the buffers that are received from the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
ToHost	Indicates the number of bytes of FILLER that follows this data item in the buffers that are sent to the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
Scale	The parameter scale.
SignAttribute	Parameter sign attribute. The possible values are: • **Trailing**. For signed DISPLAY data type, indicates that the sign is trailing (default). This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Trailing Separate**. For signed DISPLAY data type, indicates that the sign is separate. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Leading**. For signed DISPLAY data type, indicates that the sign is leading. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Leading Separate**. For signed DISPLAY data type, indicates that the sign is separate.
Size	Specifies the length of the string.

SOSI	Specifies whether double-byte character set data is expected to begin with a shift-out (SO) and end with a shift-in (SI) character. When this check box is selected, the SO and SI characters are removed from the data when it is received from the host application, and the SO and SI characters are added to the data when it is sent to the host application. In the length of the PIC G, it is not necessary to include the two bytes for the SO and SI characters because the TI run-time environment applies them. The possible values are: • **True** • **False** (default)
StringDelimiting	Return value string delimiting. The possible values are: • **Space-Padded**. Tells the TI run-time environment that the mainframe representation of the string is delimited by padding the string definition with space characters. For example, if the mainframe's COBOL definition is PIC X(10) but only three characters are in the string, the mainframe expects seven trailing spaces. Therefore, selecting this option tells the TI run-time environment to convert strings being sent to the mainframe to change the string's NULL termination character to the appropriate number of trailing spaces before sending it to the mainframe. For example, if the string is defined on the mainframe as PIC X(10), TI will send a string of *ABC* followed by seven trailing spaces. Selecting this option also tells the TI run-time environment to convert the output string being returned from the mainframe to the TI Automation server by converting the string's trailing spaces to a single null termination character. (default) • **Null-terminated**. Tells the TI run-time environment that the mainframe representation of the string is delimited by a null character (EBCDIC 0x00). Selecting this option tells the TI run-time environment to add a single null character to the end of a string if there is room for the byte before sending a string to the mainframe, and it tells the TI run-time environment to stop at the first null character encountered when receiving a string from the mainframe. Therefore, by selecting this option, you are telling TI to retain trailing spaces in output strings coming from the mainframe because TI will not convert the trailing spaces to a single NULL terminator. For more information, see Padding Mainframe Character Strings with Spaces.

Design Properties

Use this	To do this

Data Type	The possible values are: • **Void** • **Boolean** • **Byte** • **Date** • **Currency** • **Decimal** • **Integer** • **Long** • **Double** • **Single** • **String** • **User-defined type** • **Recordset** • **(none)** (default)
Name	Name of the parameter. The name can be a maximum of 250 Unicode characters.
Parameter Direction	The possible values are: • **In** • **Out** • **In / Out** (default)

Recordset Properties

Use this	**To do this**
Include Actual Size	Default option indicating that the host program will not include or expect any information that indicates the actual number of rows (recordsets) or elements (arrays) being sent or received. The possible values are: • **True** • **False** (default)

Maximum Occurrence	Maximum row occurrence. Indicates the maximum number of rows to be sent to or received from the host. Equivalent to the OCCURS n TIMES keyword on a COBOL group item.
Occurs Depending On	Parameter recordset occurs depending on. Equivalent to variable-length tables in COBOL. Indicates that a numeric data item preceding the table (recordset or array in Automation) indicates the actual number of rows or elements being sent or received. Use the drop-down list to select which numeric data item specifies this value. For CICS Link, the recordset or array and the associated length specifier must be in/out. The data in the buffer that follows a variable length table immediately follows the last data item in the table regardless of the maximum size specified for the table. For arrays with multiple dimensions, it can only be used for the outermost loop (COBOL) or rightmost dimension (Visual C++ or Visual Basic).
Unbounded	Indicates the recordset is unbounded. Indicates that any number of rows can be sent to or received from the host. You would select this option when the rows being sent or received are from a database and the maximum number of rows is not known. The possible values are: • **True** • **False** (default)

Caution

The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Tasks

How to Pad Mainframe Character Strings with Spaces

Reference

Parameter Name Node (COM)

Parameter Name Node (.NET)

Recordset Properties

Use the **Recordset** properties page to set design and host definition properties on a recordset.

Design Properties

Use this	To do this
Name	Name of the recordset. The name can be a maximum of 250 Unicode characters.

Host Definition Properties

Use this	To do this
Column Filler	Recordset filler. Indicates for each row of data sent or received, the number of bytes of FILLER that precedes each row. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
Variable Sized Rows	Recordset variable sized rows. Use this option to indicate that the last data item or group item returned from the host varies from zero to the maximum size specified for the data item. When the last data item is character data, it refers to the number of bytes (TI terminology is "variably sized"). When the last data item is an array, it refers to the number of elements. When the last data item is a recordset, it refers to the number of rows. The last two are called "bounded" in TI terminology. The possible values are: • **(none)** (default) • **Half-word binary (16 bits)**. Indicates that the length specifier for the actual size of a variably sized row will be a half-word binary (16-bit) value. This is the default. • **Full-word binary (32 bits)**. Indicates that the length specifier for the actual size of a variably sized row will be a full word binary (32-bit) value. • **Half Word (16 bits) Inclusive**. If the value of the actual size of variably sized rows will include the length specifier (the actual size of the row plus 2 or 4 bytes), select this check box. Verify that this check box is cleared if the value should only specify the actual size of a row itself. • Full Word (32 bits) Inclusive.

⚠**Caution**
The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

Recordset Column Properties

Recordsets Node

Recordset Name Node

Recordset Column Properties

Use the **Recordset Column** properties page to set COBOL and design properties on a recordset column.

COBOL Properties

Use this	To do this
HostDataType	Specifies the recordset column host data type.
ErrorHandling	Recordset column error handling. The possible values are: • **Truncate**. If selected and an error occurs, TI will truncate the value. (default) • **Round**. If selected and an error occurs, TI will round the value. • **Error**. If selected and an error occurs, TI will return an error.
Filler	Recordset column filler. The default is **0**.
FromHost	Indicates the number of bytes of FILLER that follows this data item in the buffers that are received from the server. FILLER ca uses an untranslated gap in the buffer. FILLER is not visible on the Automation side.
ToHost	Indicates the number of bytes of FILLER that follows this data item in buffers that are sent to the server. FILLER causes an un translated gap in the buffer. FILLER is not visible on the Automation side.
Scale	Recordset column scale.

SignAttribute	Recordset column sign attribute. The possible values are: • **Trailing**. For signed DISPLAY data type, indicates that the sign is trailing (default). This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. For signed DISPLAY data type, indicates that the sign is not separate (default). • **Trailing Separate**. For signed DISPLAY data type, indicates that the sign is separate. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. • **Leading**. For signed DISPLAY data type, indicates that the sign is leading. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. For signed DISPLAY data type, indicates that the sign is not separate (default). • **Leading Separate**. For signed DISPLAY data type, indicates that the sign is separate.
Size/Precision	Recordset column size.
SOSI	Specifies whether double-byte character set data is expected to begin with a shift-out (SO) and end with a shift-in (SI) character. When this check box is selected, the SO and SI characters are removed from the data when it is received from the host application, and the SO and SI characters are added to the data when it is sent to the host application. In the length of the PIC G, it is not necessary to include the two bytes for the SO and SI characters because the TI run-time environment applies them. The possible values are: • **True** • **False** (default)

StringDelimiting	Recordset column string delimiting. The possible values are: • **Space-padded**. Tells the TI run-time environment that the mainframe representation of the string is delimited by padding the string definition with space characters. For example, if the mainframe's COBOL definition is PIC X(10) but only three characters are in the string, the mainframe expects seven trailing spaces. Therefore, selecting this option tells the TI run-time environment to convert strings being sent to the mainframe to change the string's NULL termination character to the appropriate number of trailing spaces before sending it to the mainframe. For example, if the string is defined on the mainframe as PIC X(10), TI will send a string of *ABC* followed by seven trailing spaces. Selecting this option also tells the TI run-time environment to convert the output string being returned from the mainframe to the TI Automation server by converting the string's trailing spaces to a single null termination character. For more information, see Padding Mainframe Character Strings with Spaces. • **Null-terminated**. Tells the TI run-time environment that the mainframe representation of the string is delimited by a null character (EBCDIC 0x00). Selecting this option tells the TI run-time environment to add a single null character to the end of a string if there is room for the byte before sending a string to the mainframe, and it tells the TI run-time environment to stop at the first null character encountered when receiving a string from the mainframe. Therefore, by selecting this option, you are telling TI to retain trailing spaces in output strings coming from the mainframe because TI will not convert the trailing spaces to a single NULL terminator. For more information, see Padding Mainframe Character Strings with Spaces.

Design Properties

Use this	To do this
Data Type	The data type of the currently displayed column. Recordset column data type. The possible values are: • **Void** • **Boolean** • **Byte** • **Date** • **Currency** • **Decimal** • **Integer** • **Long** • **Double** • **Single** • **String**
Name	Name of the recordset column. The name can be a maximum of 250 Unicode characters.

Caution

The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

Recordset Column Name Node

User-Defined Type Properties

Use the **User-Defined Type** properties page to set design and host definition properties on user-defined types (UDTs).

Design properties

Use this	To do this
Name	Name of the user-defined type (UDT). The name can be a maximum of 250 Unicode characters.

Host Definition properties

Use this	To do this
Independent UDT	Independent user-defined type. An independent user-defined type (UDT) is a UDT that is not referenced by a method (directly or indirectly). When you use TCP/IP, the client sends the host a transaction request message (TRM) or enhanced listener message (ELM) containing the Transaction Program ID, User ID, Password, and other administrative data to be used by the host. The client sends a TRM or ELM reply containing additional administrative data. The data in the TRM or ELM is independent from the actual program data to be exchanged with the Transaction Program on the host. You can use the independent UDT options and the naming convention of TRMIN, TRMOUT, ELMIN, or ELMOUT to control the data content and format in the TRM or ELM request and TRM or ELM reply. For TRMs or ELMs destined for the host, the name of the UDT must begin with the characters TRMIN or ELMIN. For TRM or ELM replies from the host, the name of the UDT must begin with the characters TRMOUT or ELMOUT. Examples of valid TRM names are: TRMINMyVeryOwn, ELMINStandard, TRMOUTMyVeryOwn, and ELMOUTStandard. When you begin the UDT with the character TRMIN, TRMOUT, ELMIN, or ELMOUT, Visual Studio automatically formats the first member as an Int or Long and the last members as a String or Array. After an independent UDT has been defined, it can be referenced by the client application and passed to and from the TI runtime (by using the COMTIContext object) as an optional parameter. The possible values are: • **(none)** (default) • **Length Inclusive** • **Length Exclusive**
Member Filler	User-defined type member filler. Type the number of bytes of FILLER that precedes each row of data sent or received. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side. This option is not available if the length specifier option to include or exclude itself is selected.

⚠Caution
The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

User-Defined Types Node

User-Defined Type Name Node

User-Defined Type Member Properties

Use the **User-Defined Type Member** properties page to set array, COBOL, host, design, and recordset properties on user-defined type members.

Array Properties

Use this	To do this
Array Dimensions	User-defined type member array dimensions. The default is **(none)**.
Is Array	User-defined type member is an array. The possible values are: • **True** • **False** (default)
Occurs Depending On	User-defined type member array occurs depending on.

Host Properties

Use this	To do this
HostDataType	User-defined type member host data type.
ErrorHandling	User-defined type member error handling. The possible values are: • **Truncate**. If selected and an error occurs, TI will truncate the value. (default) • **Round**. If selected and an error occurs, TI will round the value. • **Error**. If selected and an error occurs, TI will return an error.
Filler	User-defined type member filler.
FromHost	Indicates the number of bytes of FILLER that follows this data item in the buffers that are received from the server. FILLER ca uses an untranslated gap in the buffer. FILLER is not visible on the Automation side.

ToHost	Indicates the number of bytes of FILLER that follows this data item in buffers that are sent to the server. FILLER causes an untranslated gap in the buffer. FILLER is not visible on the Automation side.
Scale	User-defined type member scale.
SignAttribute	User-defined type member sign attribute. The possible values are: • **Trailing**. For signed DISPLAY data type, indicates that the sign is trailing (default). This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. For signed DISPLAY data type, indicates that the sign is not separate (default). • **Trailing Separate**. For signed DISPLAY data type, indicates that the sign is separate. • **Leading**. For signed DISPLAY data type, indicates that the sign is leading. This option indicates to the TI run-time environment how a signed DISPLAY data type is formatted and affects how data from the host is converted to and from the Automation data type. For signed DISPLAY data type, indicates that the sign is not separate (default). • **Leading Separate**. For signed DISPLAY data type, indicates that the sign is separate.
Size	User-defined type member size.
SOSI	Specifies whether double-byte character set data is expected to begin with a shift-out (SO) and end with a shift-in (SI) character. When this check box is selected, the SO and SI characters are removed from the data when it is received from the host application, and the SO and SI characters are added to the data when it is sent to the host application. In the length of the PIC G, it is not necessary to include the two bytes for the SO and SI characters because the TI run-time environment applies them. The possible values are: • **True** • **False** (default)

StringDelimiting	User-defined type member string delimiting. The possible values are: • **Space-padded**. Tells the TI run-time environment that the mainframe representation of the string is delimited by padding the string definition with space characters. For example, if the mainframe's COBOL definition is PIC X(10) but only three characters are in the string, the mainframe expects seven trailing spaces. Therefore, selecting this option tells the TI run-time environment to convert strings being sent to the mainframe to change the string's NULL termination character to the appropriate number of trailing spaces before sending it to the mainframe. For example, if the string is defined on the mainframe as PIC X(10), TI will send a string of *ABC* followed by seven trailing spaces. Selecting this option also tells the TI run-time environment to convert the output string being returned from the mainframe to the TI Automation server by converting the string's trailing spaces to a single null termination character. For more information, see Padding Mainframe Character Strings with Spaces. • **Null-terminated**. Tells the TI run-time environment that the mainframe representation of the string is delimited by a null character (EBCDIC 0x00). Selecting this option tells the TI run-time environment to add a single null character to the end of a string if there is room for the byte before sending a string to the mainframe, and it tells the TI run-time environment to stop at the first null character encountered when receiving a string from the mainframe. Therefore, by selecting this option, you are telling TI to retain trailing spaces in output strings coming from the mainframe because TI will not convert the trailing spaces to a single NULL terminator. For more information, see Padding Mainframe Character Strings with Spaces.

Design Properties

Use this	To do this
Data Type	User-defined type member data type. The data type of the currently displayed user-defined type member. The possible values are: • **Void** • **Boolean** • **Byte** • **Date** • **Currency** • **Decimal** • **Integer** • **Long** • **Double** • **Single** • **String** • **User-defined type** • **Recordset**

Name	Name of the user-defined type member. The name can be a maximum of 250 Unicode characters. The name must be unique from any other user-defined type member name in the same project. The default is **null**.

Recordset Properties

Use this	To do this
Include Actual Size	Default option indicating that the host program will not include or expect any information that indicates the actual number of rows (recordsets) or elements (arrays) being sent or received. The possible values are: • **True** • **False** (default)
Maximum Occurrence	Maximum row occurrence. Indicates the maximum number of rows to be sent to or received from the host. Equivalent to the OCCURS n TIMES keyword on a COBOL group item.
Occurs Depending On	User-defined type member recordset occurs depending on. Indicates the maximum number of rows to be sent to or received from the host. Equivalent to the OCCURS n TIMES keyword on a COBOL group item. Equivalent to variable-length tables in COBOL. Indicates that a numeric data item preceding the table (recordset or array in Automation) indicates the actual number of rows or elements being sent or received. Use the drop-down list to select which numeric data item specifies this value. For CICS Link, the recordset or array and the associated length specifier must be in/out. The data in the buffer that follows a variable length table immediately follows the last data item in the table regardless of the maximum size specified for the table. For arrays with multiple dimensions, it can only be used for the outermost loop (COBOL) or rightmost dimension (Visual C++ or Visual Basic).
Unbounded	Indicates that any number of rows can be sent to or received from the host. Set to **true** when the rows being sent or received are from a database and the maximum number of rows is not known. The possible values are: • **True** • **False** (default)

⚠Caution

The properties of a component are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the component to function incorrectly.

See Also

Reference

User-Defined Type Member Name Node

Unions Properties

Displays properties for the Unions folder.

Union Type Properties

Displays the name of the Union type.

Union Member Properties

Lists the properties of the Union Member. Depending on the Host Data Type, the list may include:

- Array Dimensions
- Data Types
- Error Handling
- Host Data Type
- Is Array
- Name
- Precision
- Scale
- Size
- String Delimiting
- SOSI
- Trailing Filler

Name Conflict Dialog Box

Use the **Name Conflict** dialog box to rename what you are copying.

Use this	To do this
New name	Type a new name. The name can be a maximum of 256 Unicode characters. The new name cannot be the same as and existing name.

See Also

Reference

Interface Properties

Array Dimension Dialog Box

Use the **Array Dimension** dialog box to define the number of dimensions in an array and the number of elements in each dimension.

Use this	To do this
Number of Dimensions	Type the number of dimensions in the array. The minimum number is 0, and the maximum is 7. The default is **0**.
Dimension/Number of Elements	View the current number of elements defined for each dimension.
Number of Elements	Type the number of elements in the selected dimension. The minimum number is 1, and the maximum number is 16777215. The default is **10**.

See Also

Reference

Parameter Properties

Map Remote Environment Class Dialog Box

Use the **Map Remote Environment Class** dialog box to update the definition of a remote environment (RE) class that is no longer supported or does not conform to Transaction Integrator (TI) requirements.

Use this	To do this
Remote environment class	Type or select the COM type library or .NET assembly class of the remote environment.
Vendor	Select the name of the vendor supplying the remote environment.
Protocol	Select the data communication protocol to be used to connect to the remote environment. The available communication protocols are: • **TCP/IP** (default) • **LU 6.2**
Target environment	Select the application environment running on the remote host system. The available environments are: • **CICS** (default) • **IMS** • **OS400** (TCP only)
Programming model	Select the programming model associated with the remote environment. The available models for the TCP/IP CICS target environment are: • **TRM User Data** (default) • **TRM Link** • **ELM User Data** • **ELM Link** The available models for the LU 6.2 CICS target environment are: • **CICS User Data** • **CICS Link** • The available models for the TCP/IP IMS target environment are: • **IMS Connect** • **Implicit** • **Explicit** • The only available model for the LU 6.2 IMS target environment is **IMS User Data**. The only available model for the OS400 target environment is **Distributed Program Call**.

See Also

Other Resources

Wizards and Dialog Boxes (TI Project)

Select Convert Prim Dialog Box

Use the **Select Convert Prim** dialog box to provide a Convert PrimEx class that is missing from your server library.

Use this	To do this
Convert prim class	Select the conversion class to use with the host environment (HE). The available classes are: • **Transaction Integrator - ConvertPrim for OS390** (default) • **Transaction Integrator - ConvertPrim for AS400**

Host Integration Server Designer UI

Host Integration Server Designer (HIS Designer) is a graphical user interface for creating Transaction Integrator (TI) components, which are annotated type libraries or assemblies. You can use HIS Designer to export or import the COBOL data declarations used in mainframe CICS and IMS programs. HIS Designer is a standalone program that does not need to have connectivity to the mainframe.

HIS Designer is hosted within the Visual Studio environment. You can use it to generate both Windows-initiated processing (WIP) and host-initiated processing (HIP) objects. It also supports COM type libraries and Microsoft .NET Framework assemblies.

In This Section

See Also

Concepts

Getting Started with TI

Solution Explorer

Visual Studio Solution Explorer provides an organized view of projects and their files, in addition to access to available commands and toolbars. The following files are supported by the project:

- COM Client-Initiated libraries, which are annotated COM type libraries (.tlb).
- .NET Client-Initiated libraries, which are Microsoft .NET Framework assemblies with embedded annotated COM type library.
- Host-Initiated libraries, which are annotated COM type libraries (.tim).

To open Solution Explorer

- On the **View** menu, click **Solution Explorer**.

See Also
Concepts
Host Integration Server Designer UI

Add New Item Dialog Box (Visual Studio)

The Add New Item dialog box enables you to add an item to the currently selected project. There are two panes in the dialog box:

- The **Categories** pane lists the project item hierarchy.
- The **Templates** pane lists the related project item types.

When you select an item from the **Categories** list, the appropriate files and references are added to your project. The type of project selected determines the item choices that are displayed. The project items created for Transaction Integrator include the following:

- COM Server Library, which is an annotated COM type library that you must implement separately, and the implemented object, which can be invoked from an IBM host.
- COM Client Library, which is a COMTI classic type library that can call host applications.
- .NET Server Library, which is an assembly with an embedded COM Server library. It is a managed version of the COM Server library.
- .NET Client Library, which is an assembly with an embedded COM Client library. It is a managed version of the COM client library.

To add a new item

1. On the **Project** menu, click the type of library (COM client, COM Server, .NET Client, or .NET Server) you want to add.
2. On the **Add New Item** dialog box, confirm that the correct template is highlighted.
3. Next to **Name**, type the name that you want to use for the library.
4. Click **Add**, and follow the directions for the relevant Wizard.

See Also
Concepts
Host Integration Server Designer UI

New COM Client Library Wizard

The **New COM Client Library Wizard** is a two-page wizard. The first page of the wizard collects library information, and the second page collects remote environment information. There is a **Welcome** page and a **Finish** page that displays summary information.

The generated client component is either an annotated COM type library (.tlb) or a Microsoft .NET Framework assembly with the annotated type library embedded in it as resources; it depends on the platform that you select in the Add New Item Dialog Box (Visual Studio).

When the file is created, it is added to the project and displayed in Solution Explorer.

See Also

Concepts

Host Integration Server Designer UI

New COM or .NET Server Library Wizard

Like the **New COM or .NET Client Library Wizard**, this wizard is a two-page wizard. The first page collects library information, and the second page collects host environment information. In addition, there is a **Welcome** page, and a **Finish** page that displays summary information.

The generated client component is either an annotated COM type library (.tlb) or a Microsoft .NET Framework assembly with the annotated type library embedded in it as resources; it depends on the platform that you selected in the Add New Item Dialog Box (Visual Studio). When the file is created, it is added to the project and displayed in Solution Explorer.

See Also
Concepts
Host Integration Server Designer UI

Import Library Wizard

The **Import Library Wizard** enables you to import an existing library, whether it is annotated or not, to the existing component. The methods, recordsets, user-defined types, and unions from the imported library are inserted under the library in the Host Integration Server (HIS) designer.

You can start this wizard by selecting **Import Library** on the **Library** top-level menu. This wizard does not have welcome and finish page because it only selects a file to import. It is a single page dialog box, which is the standard Open File dialog box.

See Also

Concepts

Host Integration Server Designer UI

Welcome to the Import COBOL Wizard

The **Import COBOL Wizard** enables you to select a COBOL copy book and use it for designing the component in the designer. You can start this wizard by selecting **Import COBOL** on the **Library** top-level menu.

This is a rather complex wizard because there are multiple options and paths to follow. It is a multiple-page wizard that contains **Welcome** and **Finish** pages.

See Also
Concepts
Host Integration Server Designer UI

Export Wizard

The **Export Wizard** is used in exporting an equivalent COBOL definition for the current component. It is not really a wizard because it does not contain any pages, such as a welcome or finish page. You can start this wizard by selecting **Export** on the **Library** top-level menu. When you select the menu item, a text window opens under Visual Studio .NET, and the COBOL is displayed in it. You can then save it, discard it, or add it to the project.

See Also
Concepts
Host Integration Server Designer UI

HIS Designer Options

You can select the initial view to display when the Host Integration Server Designer (HIS Designer) starts. You can modify this option through the standard **Options** dialog box in Visual Studio, which is extended to support the functionality of the HIS Designer. You can open this dialog box by clicking **Options** on the **Tools** menu in Visual Studio.

See Also
Concepts
Host Integration Server Designer UI

HIS Designer Views

The Host Integration Server Designer (HIS Designer) development tool uses a two-pane user interface. The left pane displays a tree view, and the right pane displays a list view, or **Details** view. The tree view displays a hierarchical representation of the contents of the library. The list view displays a compact subset of the selected type library component's properties that are displayed in the property browser. You might find it useful to customize the details view by using something other than a list. For example, a COBOL, RPG, or IDL view might be more useful.

Tree View

The tree view displays a hierarchical view of the components in a type library or Microsoft .NET Framework assembly.

The tree view contains the following elements:

Type Library

- Methods
    - Method1
    - Parameter1
    - Parameter2
    - ...
    - Parameter(n)
    - Method2
    - ...
    - Method(n)
- Recordsets (Data tables for .NET Framework libraries)
    - Recordset1
    - Recordset Member1
    - Recordset Member2
    - ...
    - Recordset Member(n)
    - Recordset2
    - ...
    - Recordset(n)
- User-Defined types (Structures for .NET Framework libraries)
    - UDT1
    - UDT Member1

        - UDT Member2
        - ...
        - UDT Member(n)
        - UDT2
        - ...
        - UDT(n)
- Unions
        - Union1
        - Union Member1
        - Union Member2.
        - ...
        - Member(n)
        - Union2
        - ...
        - Union(n)

Details View

The details view displays context-sensitive information about the item that is selected in the tree view. The details view displays a subset of the properties of that type library component. These properties are displayed in columns labeled in a context-sensitive manner.

The detail views include the following:

- Interface view
- Method view
- Parameter view
- Recordsets view
- Recordset view
- Recordset column view
- User-defined types view
- User-defined type view
- User-defined type member view

- Unions view
- Union view
- Union member view

See Also
Concepts
Host Integration Server Designer UI

HIS Designer Menus

Host Integration Server Designer (HIS Designer) presents different menus for different contexts and operations. These menus are in addition to the basic menus that are already included in Visual Studio.

In This Section

HIS Designer Main Menu

HIS Designer Shortcut Menus

Importing RPG

See Also

Concepts

Host Integration Server Designer UI

HIS Designer Main Menu

The main menu in Host Integration Server Designer (HIS Designer) presents two tabs, **Edit** and **Library**, which contain individual commands.

Edit Tab

The following commands appear on the **Edit** tab of the main menu:

Undo: Undo the last operation.

Redo: Redo the last undo operation.

Cut: Cut the selected item.

Copy: Copy the selected item.

Paste: Paste the copied item to the selected one.

Rename: Rename the selected item.

Delete: Delete the selected item.

Select All: Select all items.

Find and Replace: Find and replace the specified text.

Move Up: Move a parameter up in the list.

Move Down: Move a parameter down in the list.

Library Tab

The following commands appear on the **Library** tab of the main menu:

Add Method: Add a new method with an integer return type.

Add Parameter: Add a new In\Out integer parameter to the selected method. This command is only visible if the current selection in the tree view or the details view is a method.

Add Recordset: Add a new recordset with an integer column.

Add Recordset Column: Add a new integer recordset column to the selected recordset. This command is only visible if the current selection in the tree view or the details view is a recordset.

Add User-Defined Type: Add a new user-defined type with an integer member.

Add Union: Add a new union with two integer members.

Add Union Member: Add a new integer union member.

Add User-Defined Type Member: Add a new integer user-defined member to the selected user-defined type. This command is only visible if the current selection in the tree view or the details view is a user-defined type.

Import Host Definition: Invoke the Import COBOL Wizard, which enables you to import COBOL to the interface (Class) definition.

Import Library: Invoke the Library Import Wizard, which enables you to import an existing type library or assembly to the interface (Class) definition.

Export Host Definition: Invoke the Export Wizard, which enables you to generate a COBOL copy book equivalent to the current library.

Definition: Display the definition of the current library in a new Visual Studio default editor window. With COM libraries, you can display the IDL definition, and with .NET Framework assemblies, you can display the Cdefinition. When the definition is available in Visual Studio, all standard file operations can be applied to it (for example, **Save**, **Save As**, and so on).

Host Data Definition:

See Also
Concepts
HIS Designer Menus

HIS Designer Shortcut Menus

Host Integration Server Designer (HIS Designer) has two types of context-sensitive, or shortcut menus. The first type is context-sensitive to the HIS Designer files in Visual Studio Solution Explorer. The second type is context-sensitive in the HIS Designer itself.

Shortcut menus do not have context-sensitive Help available when you use them. Therefore, the topics in this section describe the functionality of each menu and each command.

In This Section

Solution Explorer Shortcut Menu

When you select a Transaction Integrator-compatible file in Solution Explorer, the appropriate context-sensitive, or shortcut menu commands are displayed.

Add: Adds a new item to the project.

Method: Adds a new method with an integer return type.

Parameter:

Recordset: Adds a new recordset with an integer column.

Recordset Member:

User-defined Type: Adds a new user-defined type with an integer member.

User-defined Type Member:

Import: Imports an existing item into the project.

COBOL: Starts the **Import COBOL Wizard** to help you import COBOL to the interface (Class) definition.

Library: Starts the **Library Import Wizard** to help you import an existing type library or assembly to the interface (Class) definition.

Export: Starts the **Export Wizard** to help you generate a COBOL copy book equivalent to the current type library.

Generate Definition: Displays the definition of the current library. For COM libraries, it displays the IDL definition; for .NET Framework assemblies, it displays the C# definition.

See Also

Concepts

HIS Designer Shortcut Menus

Library Shortcut Menu

When you select the **Library** node in the tree view of HIS Designer, the following shortcut menu commands appear.

Import: Displays the following commands:

Host Definition: Starts the **Import COBOL Wizard** to help you import COBOL to the interface (Class) definition.

HCD: Imports a Host Column descriptor. Usable only with a Host File Project.

Library: Starts the Library Import Wizard to help you import an existing type library or assembly to the interface (Class) definition.

Export: Starts the **Export Wizard** to help you generate a COBOL copy book equivalent to the current library.

Lock: Marks the library as read-only. The library is automatically marked as locked if it is registered in a COM+ application, IIS Virtual Directory, or is being used by a host-initiated processing (HIP) application.

Deploy: Deploys the library.

Undeploy: Undeploys the library.

Cut: Copies the content of the library to the clipboard and marks it as deleted.

Copy: Copies the content of the library to the clipboard.

Paste: Inserts the contents of the clipboard into the current library definition.

Delete: Deletes the current library.

Rename: Renames the library.

Properties: Displays the property browser (if it is not already visible) and displays the library properties.

See Also

Concepts

HIS Designer Shortcut Menus

Interface Shortcut Menu

When you select the **Interface** node in the tree view of HIS Designer, the following shortcut menu commands are available:

Add Method: Adds a method to the interface.

Cut:. Visible, but not available.

Copy: Visible, but not available.

Paste: Visible, but not available.

Rename: Renames the interface.

Delete: Visible, but not available.

Properties: Displays the property browser (if it is not already visible) and displays the interface properties.

See Also

Concepts

HIS Designer Shortcut Menus

Method Shortcut Menu

When you select a method in the tree view in HIS Designer, the following context menu commands are available.

Add Parameter: Adds a new In\Out integer parameter to the selected method.

Cut: Copies the selected method to the clipboard and marks it as deleted.

Copy: Copies the selected method to the clipboard.

Paste: Inserts a parameter from the clipboard into the current method.

Delete: Deletes the method.

Rename: Renames the method.

Properties: Displays the property browser (if it is not already visible) and displays the method properties.

See Also
Concepts
HIS Designer Shortcut Menus

Parameter Shortcut Menu

When you select a parameter in the tree view in HIS Designer, the following shortcut menu commands are displayed.

Move Up: Moves the selected parameter up in the tree list.

Move Down: Moves the selected parameter down in the tree list.

Cut: Copies the selected parameter to the clipboard and marks it as deleted.

Copy: Copies the selected parameter to the clipboard.

Paste: Not available because parameters do not have child elements.

Delete: Deletes the parameter.

Rename: Renames the parameter.

Properties: Displays the property browser (if it is not already visible) and displays the parameter properties.

See Also
Concepts
HIS Designer Shortcut Menus

DataTables Shortcut Menu

When you select the **DataTables** node from the library tree, the following shortcut menu commands are displayed:

Add DataTable: Adds a DataTable to the DataTables node.

Cut: Visible, but not available.

Copy: Visible, but not available.

Paste: Visible, but not available.

Delete: Visible, but not available.

Rename: Visible, but not available..

Properties: Displays the property browser (if it is not already visible) and displays the DataTables properties.

See Also

Concepts

HIS Designer Shortcut Menus

DataTable Shortcut Menu

When you select a **DataTable** node in the tree view, the following shortcut menu commands are displayed.

Add DataTablet Column: Adds a new integer column to the selected DataTable.

Cut: Copies the selected DataTable to the clipboard and marks it as deleted.

Copy: Copies the selected DataTable to the clipboard.

Paste: Inserts a DataTable column from the clipboard into the current DataTable.

Delete: Deletes the DataTable.

Rename: Renames the DataTable.

Properties: Displays the property browser (if it is not already visible) and displays the DataTable properties.

See Also

Concepts

HIS Designer Shortcut Menus

DataTable Column Shortcut Menu

When you select the **DataTable** column in the tree view, the following shortcut menu items are displayed.

Move Up: Moves the selected column up in the tree list.

Move Down: Moves the selected column down in the tree list.

Cut: Copies the selected DataTable column to the clipboard and marks it as deleted.

Copy: Copies the selected DataTable column to the clipboard.

Paste: Not available because DataTable columns do not have child elements.

Delete: Deletes the DataTable column.

Rename: Renames the DataTable column.

Properties: Displays the property browser (if it is not already visible) and displays the DataTable column properties.

See Also

Concepts

HIS Designer Shortcut Menus

Structures Shortcut Menu

When you select the **Structures** node in the library tree, the following shortcut menu commands appear:

Add Struct: Adds a structure to the selected Structure node.

Cut: Visible, but not available.

Copy: Visible, but not available.

Paste: Visible, but not available.

Rename: Visible, but not available.

Delete: Deletes the recordset column.

Properties: Displays the property browser (if it is not already visible) and displays the Structure column properties.

See Also

Concepts

HIS Designer Shortcut Menus

Structure Shortcut Menu

When you select a structure type in the tree view in HIS Designer, the following shortcut menu commands are displayed.

Add Structure Member: Adds a new integer member to the selected structure.

Cut: Copies the selected structure to the clipboard and marks it as deleted.

Copy: Copies the selected structure to the clipboard.

Paste: Inserts a structure column from the clipboard into the current structure.

Delete: Deletes the structure.

Rename: Renames the structure.

Properties: Displays the property browser (if it is not already visible) and displays the structure properties.

See Also
Concepts
HIS Designer Shortcut Menus

Structure Member Shortcut Menu

When you select a member from one of the structures in the **Structures** node in HIS Designer, the following shortcut menu commands are available:

Move Up: Moves the selected member up in the tree list.

Move Down: Moves the selected member down in the tree list.

Cut: Copies the selected member to the clipboard and marks it as deleted.

Copy: Copies the selected member to the clipboard.

Paste: Not available because members do not have child elements.

Delete: Deletes the member.

Rename: Renames the member.

Properties: Displays the property browser (if it is not already visible) and displays the member properties.

See Also

Concepts

HIS Designer Shortcut Menus

Unions Shortcut Menu

When you select the **Unions** node in the tree view in HIS Designer, the following shortcut menu commands are displayed.

Add Union: Adds a union to the **Unions** node.

Cut: Visible but not available.

Copy: Visible but not available.

Paste: Visible but not available.

Delete: Visible but not available.

Rename: Visible but not available.

Properties: Displays the property browser (if it is not already visible) and displays the Unions properties.

See Also

Concepts

HIS Designer Shortcut Menus

Union Shortcut Menu

When you select a union from the **Union** node in the **Library** tree, the following shortcut menu commands are available:

Add Union Member: Adds a union union to the selected union.

Cut: Copies the selected union to the clipboard and marks it as deleted.

Copy: Copies the selected union to the clipboard.

Paste: Not available because unions do not have child elements.

Delete: Deletes the union.

Rename: Renames the union.

Properties: Displays the property browser (if it is not already visible) and displays the union properties.

See Also

Concepts

HIS Designer Shortcut Menus

Union Member Shortcut Menu

When you select a union member from the Library tree, the following shortcut menu commands appear:

Move Up: Moves the selected union member up in the tree list.

Move Down: Moves the selected union member down in the tree list.

Cut: Copies the selected union member to the clipboard and marks it as deleted.

Copy: Copies the selected union member to the clipboard.

Paste: Not available because union members do not have child elements.

Delete: Deletes the union member.

Rename: Renames the union member.

Properties: Displays the property browser (if it is not already visible) and displays the union member properties.

See Also

Concepts

HIS Designer Shortcut Menus

Importing RPG

The Report Program Generator (RPG) is oriented to programs written for use by distributed program call (DPC). The expectation is that the RPG source contains an ***ENTRY PLIST** operation statement. This statement leads the importer down the path of explicitly understanding the exact parameter names. This eliminates the picking and choosing of data areas found in the **Import COBOL Wizard**.

This implies that the RPG Importer is simple to use. It also implies that the RPG Importer is not designed to be used for raw TCP and raw SNA programming models. The assumptions are as follows:

- RPG DPC is the most likely programming model that will be used.
- For those situations where raw TCP or SNA is required, you can still create RPG definitions through basic HIS Designer functionality (not the Import Wizard)

Discriminant Value Table Dialog Box

Use the **Discriminant Value Table** dialog box to create the logic for determining which union member returns information to the calling procedure. You can open the **Discriminant Value Table** dialog box by viewing the properties of an instanced union member, clicking **DVT**, and then clicking the ellipsis (**...**) button.

Use this	To do this
Discriminant Variable	List the name of the selected union.
Discriminant type	List the data type that the union will use to create the logic.
Discriminant Value Table	Create the Discriminant Value Table. The **Union Member** column lists the union members that will be checked, in order. The **Condition** column describes what to check for in order to use that member.
Move Up	Move a union member and associated condition up in priority.
Move Down	Move a union member and associated condition down in priority.
Delete	Delete a union member and associated condition from the DVT. It does not delete the member from the union. You may add the union member back into the DVT at any time.
OK	Save the additions you make to the DVT, and close the dialog box.
Cancel	Close the dialog box without saving the changes you made to the DVT.

See Also

Concepts

Host Integration Server Designer UI

Host File Designer UI

Host File Designer is a graphical user interface for creating Host File components, which are annotated type libraries or assemblies. As with Host Integration Server (HIS) Designer, you can use Host File Designer export or import the COBOL data declarations used in mainframe CICS and IMS programs. Host File Designer is a standalone program that does not need to have connectivity to the mainframe.

Host File Designer is hosted within the Visual Studio environment. You can use it to generate both Windows-initiated processing (WIP) and host-initiated processing (HIP) objects.

In This Section

See Also

Other Resources

Visual Studio Help

Add New Item Dialog Box (Host File Designer)

The Add New Item dialog box enables you to add a host file object the currently selected project. There are two panes in the dialog box:

- The **Categories** pane lists the project item hierarchy.

  For Host File Designer, the only item in this pane is **Host Integration Project Items**.

- The **Templates** pane lists the related project item types.

  For Host File Designer, the only usable item in this pane is **Host Files Library**. A host files library is a .NET assembly that describes the host file system to your application.

To add a new item

1. On the **Project** menu, click **Add Host File Library** or **Add New Item...**.

   For the purposes of a Host File Designer project, you can add only a Host File Library.

2. On the **Add New Item** dialog box, confirm that **Host File Library** is highlighted.

3. Next to **Name**, type the name that you want to use for the library.

4. Click **Add**, and follow the directions for the relevant Wizard.

See Also
Concepts
Host File Designer UI

Welcome to the Host Files Library Wizard

The **Import Host Column Definition Wizard** enables you to select an .hcd file and use it for designing the component in the designer. You can start this wizard by selecting **Import HCD** on the **Library** top-level menu.

See Also
Concepts
Host File Designer UI

Host Environment (Host File Designer)

Use the **Host Environment** Wizard page to select the host environment for your host file application.

Use this	To do this
Host environment	Select the host environment. Your current choices are as follows: Host Files for OS390 Host Files for AS400

See Also

Concepts

Host File Designer UI

Completing the Host Files Library Wizard Page

Use the **Completing the Host Files Library Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

Use this	To do this
Run this wizard again	Select this option to complete the current modification of the type library and automatically restart the wizard to make new modifications.

See Also

Concepts

Host File Designer UI

Import HCD Source File Wizard Page

Use the **Import COBOL Source File** wizard page to identify the COBOL source file.

Use this	To do this
HCD file	Select the full path to the COBOL copy book to be imported.
HCD source	View the host column description source code contained in the file identified in .hcd file.

See Also

Concepts

Host File Designer UI

Schemas Wizard Page (Host Files Library)

The Schemas wizard page allows you to select the schema(s) you would like to import from the .hcd file selected in the previous wizard page.

Use this	To do this
Schemas	To select the schemas and associated schema members.

See Also

Concepts

Host File Designer UI

Solution Explorer (Host File Designer)

Visual Studio Solution Explorer provides an organized view of projects and their files, in addition to access to available commands and toolbars. The following files are supported by the project:

- Host File libraries, which are Microsoft .NET Framework assemblies that contain an interface description of the remote server.

To open Solution Explorer

On the **View** menu, click **Solution Explorer**.

See Also
Concepts
Host File Designer UI

Host File Designer Views

The Host File Designer development tool uses a two-pane user interface. The left pane displays a tree view, and the right pane displays a list view, or **Details** view. The tree view displays a hierarchical representation of the contents of the library. The list view displays a compact subset of the selected type library component's properties that are displayed in the property browser. You might find it useful to customize the details view by using something other than a list. For example, a COBOL, RPG, or IDL view might be more useful.

Tree View

The tree view displays a hierarchical view of the components in a type library, also known as a metadata assembly.

The tree view contains the following elements:

Library

- Tables
    - Table1
    - Table Member1
    - Table Member2
    - ...
    - Table Member(n)
    - Table2
    - ...
    - Table(n)
- Schemas
    - Schema1
    - Schema Member1
    - Schema Member2
    - ...
    - Schema Member(n)
    - Schema2
    - ...
    - Schema(n)
- Unions
    - Union1
    - Union Member1

- Union Member2.
- ...
- Member(n)
- Union2
- ...
- Union(n)

Details View

The details view displays information about the item that is selected in the tree view. The Details view consists of three tabs, each describing the information in the Tree view in a different fashion.

List Tab

The List tab displays a subset of the properties of the selected type library component.

The detail views include the following:

- Tables view
- Table Member view
- Schemas view
- Schema view
- Unions view
- Union view
- Union member view

Definition Tab

The Definition tab displays the type library as a .NET interface.

HCD Tab

The HCD tab displays the host column description associated with the type library.

See Also
Concepts
Host File Designer UI

Host File Designer Menus

Host File designer presents different menus for different contexts and operations. These menus are in addition to the basic menus that are already included in Visual Studio.

In This Section

Host File Designer Main Menu

Host File Designer Shortcut Menus

Host File Designer Shortcut Menus

See Also

Concepts

Host File Designer UI

Host File Designer Main Menu

The main menu in Host File Designer presents the **Edit** tab, which contain the following commands:

Undo: Undo the last operation.

Redo: Redo the last undo operation.

Cut: Cut the selected item.

Copy: Copy the selected item.

Paste: Paste the copied item to the selected one.

Rename: Rename the selected item.

Delete: Delete the selected item.

Select All: Select all items.

Find and Replace: Find and replace the specified text.

Move Up: Move a parameter up in the list.

Move Down: Move a parameter down in the list.

See Also

Concepts

HIS Designer Menus

Host File Designer Shortcut Menus

This section describes the Host File Designer Shortcut Menus for Visual Studio.

In This Section

Host File Designer Shortcut Menus

Library Shortcut Menu (Host File Designer)

Schema Member Shortcut Menu

Schema Shortcut Menu

Schemas Shortcut Menu

Solution Explorer Shortcut Menu (Host File Designer)

Table Member Shortcut Menu

Table Shortcut Menu

Tables Shortcut Menu

Union Member Shortcut Menu (Host File Designer)

Union Shortcut Menu (Host File Designer)

Unions Shortcut Menu (Host File Designer)

See Also

Concepts

Host File Designer Menus

Solution Explorer Shortcut Menu (Host File Designer)

When you select a Transaction Integrator-compatible file in Solution Explorer, the appropriate context-sensitive, or shortcut menu commands are displayed.

Add: Adds a new item to the project.

Add Host File Library: Brings up the Add New Item dialog.

New Item...:Adds a new item to the project.

Existing Item...: Adds an existing item to the project.

Cut: Cuts the current object to the clipboard and marks the object for deletion.

Paste: Pastes the object currently in the clipboard to the current location.

Open: Opens the selected object using Visual Studio

Open With: Opens the selected object with the specified application.

Remove: Removes the selected object from the Solution.

Rename: Renames the selected object.

Properties: Displays the property browser (if it is not already visible) and displays the selected object properties.

Library Shortcut Menu (Host File Designer)

When you select the **Library** node in the tree view of Host File Designer, the following shortcut menu commands appear.

Import: Displays the following commands:

Host Definition: Starts the **Import COBOL Wizard** to help you import COBOL to the interface (Class) definition.

HCD: Imports a Host Column descriptor.

Library: Starts the Library Import Wizard to help you import an existing type library or assembly to the interface (Class) definition.

Export HCD: Starts the **Export Wizard** to help you generate a COBOL copy book equivalent to the current library.

Cut: Visible but not available.

Copy: Visible but not available.

Paste: Visible but not available.

Delete: Deletes the current library.

Rename: Renames the library.

Properties: Displays the property browser (if it is not already visible) and displays the library properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Tables Shortcut Menu

When you select the **Tables** node from the **Library** tree, the following shortcut menu commands appear:

Add Table: Adds a table to the **Tables** node.

Cut: Visible but not available.

Copy: Visible but not available.

Paste: pastes a table from the clipboard into the **Tables** node.

Delete: Visible but not available.

Rename: Visible but not available.

Properties: Displays the property browser (if it is not already visible) and displays the Tables properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Table Shortcut Menu

When you select a table from the **Table** node in the **Library** tree, the following shortcut menu commands are available:

Move Up: Moves the selected table up in the tree list.

Move Down: Moves the selected table down in the tree list.

Cut: Copies the selected table to the clipboard and marks it as deleted.

Copy: Copies the selected table to the clipboard.

Paste: Visible but not available.

Delete: Deletes the table.

Rename: Renames the table.

Properties: Displays the property browser (if it is not already visible) and displays the table properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Table Member Shortcut Menu

When you select a table member from the Library tree, the following shortcut menu commands appear:

Move Up: Moves the selected table member up in the tree list.

Move Down: Moves the selected table member down in the tree list.

Cut: Copies the selected table member to the clipboard and marks it as deleted. Not available if there is only one member in the table.

Copy: Copies the selected table member to the clipboard.

Paste: Not available because table members do not have child elements.

Delete: Deletes the table member. Not available if there is only one member in the table.

Rename: Renames the table member.

Properties: Displays the property browser (if it is not already visible) and displays the table member properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Schemas Shortcut Menu

When you select the **Schemas** node in the tree view in Host File Designer, the following shortcut menu commands are displayed.

Add Schema: Adds a schema to the **Schemas** node.

Cut: Visible but not available.

Copy: Visible but not available.

Paste: Pastes a cut or copied schema into the **Schemas** node.

Delete: Visible but not available.

Rename: Visible but not available.

Properties: Displays the property browser (if it is not already visible) and displays the Schemas properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Schema Shortcut Menu

When you select a Schema from the **Schemas** node in the **Library** tree, the following shortcut menu commands are available:

When you select a union member from the Library tree, the following shortcut menu commands appear:

Move Up: Moves the selected schema up in the tree list.

Move Down: Moves the selected schema down in the tree list.

Cut: Copies the selected schema to the clipboard and marks it as deleted. Not available on schemas that were imported.

Copy: Copies the selected schema to the clipboard.

Paste: Pastes the cut or copied schema member.

Delete: Deletes the schema. Available only on schemas that are manually generated.

Rename: Renames the schema.

Properties: Displays the property browser (if it is not already visible) and displays the schema properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Schema Member Shortcut Menu

When you select a union member from the Library tree, the following shortcut menu commands appear:

Move Up: Moves the selected union member up in the tree list.

Move Down: Moves the selected union member down in the tree list.

Cut: Copies the selected union member to the clipboard and marks it as deleted.

Copy: Copies the selected union member to the clipboard.

Paste: Not available because union members do not have child elements.

Delete: Deletes the union member.

Rename: Renames the union member.

Properties: Displays the property browser (if it is not already visible) and displays the union member properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Unions Shortcut Menu (Host File Designer)

When you select the **Unions** node in the tree view in HIS Designer, the following shortcut menu commands are displayed.

Add Union: Adds a union to the **Unions** node.

Cut: Visible but not available.

Copy: Visible but not available.

Paste: Visible but not available.

Delete: Visible but not available.

Rename: Visible but not available.

Properties: Displays the property browser (if it is not already visible) and displays the Unions properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Union Shortcut Menu (Host File Designer)

When you select a union from the **Union** node in the **Library** tree, the following shortcut menu commands are available:

Add Union Member: Adds a union member to the selected union.

Cut: Copies the selected union to the clipboard and marks it as deleted.

Copy: Copies the selected union to the clipboard.

Paste: Not available because unions do not have child elements.

Delete: Deletes the union.

Rename: Renames the union.

Properties: Displays the property browser (if it is not already visible) and displays the union properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Union Member Shortcut Menu (Host File Designer)

When you select a union member from the Library tree, the following shortcut menu commands appear:

Move Up: Moves the selected union member up in the tree list.

Move Down: Moves the selected union member down in the tree list.

Cut: Copies the selected union member to the clipboard and marks it as deleted.

Copy: Copies the selected union member to the clipboard.

Paste: Not available because union members do not have child elements.

Delete: Deletes the union member.

Rename: Renames the union member.

Properties: Displays the property browser (if it is not already visible) and displays the union member properties.

See Also

Other Resources

Host File Designer Shortcut Menus

Discriminant Value Table Dialog Box (Host File Designer)

Use the **Discriminant Value Table** dialog box to create the logic for determining which union member returns information to the calling procedure. You can open the **Discriminant Value Table** dialog box by viewing the properties of an instanced union member, clicking **DVT**, and then clicking the ellipsis (**...**) button.

Use this	To do this
Discriminant Variable	List the name of the selected union.
Discriminant type	List the data type that the union will use to create the logic.
Discriminant Value Table	Create the Discriminant Value Table. The **Union Member** column lists the union members that will be checked, in order. The **Condition** column describes what to check for in order to use that member.
Move Up	Move a union member and associated condition up in priority.
Move Down	Move a union member and associated condition down in priority.
Delete	Delete a union member and associated condition from the DVT. It does not delete the member from the union. You may add the union member back into the DVT at any time.
OK	Save the additions you make to the DVT, and close the dialog box.
Cancel	Close the dialog box without saving the changes you made to the DVT.

See Also

Concepts

Host File Designer UI

Transaction Integrator Manager Help

Transaction Integrator (TI) Manager provides a graphical user interface for creating, viewing, and managing the host-initiated processing (HIP) and Windows-initiated processing (WIP) environments. The user interface is accessible through the TI Manager management console and appears in tree view, property pages, and wizards. You can display Help topics on the individual user interface controls by clicking **Help** on the TI Manager wizard page or dialog box or by selecting a control or node and pressing the **F1** key. These individual Help topics are included in this section for easier reference and review.

In This Section

- TI Manager Nodes
- TI Manager Wizards and Dialog Boxes
- TI Manager Properties

TI Manager Nodes

The Transaction Integrator (TI) Manager console provides a tree view of the primary configuration elements used in host-initiated processing (HIP) and Windows-initiated processing (WIP). HIP enables an IBM mainframe program in CICS, IMS, or MVS to call a COM or .NET object. WIP enables a Windows client application to access an IBM mainframe transaction program in CICS, IMS, or MVS.

Each major element of the process is represented by a node on the tree in the left pane of the management console. Double-clicking the node expands it; right-clicking the node displays a shortcut menu.

In This Section

Transaction Integrator (mode) Node

Use the **Transaction Integrator** (TI) node to view the basic grouping of elements within the host-initiated processing (HIP) environment and the Windows-initiated processing (WIP) environment. You can also use the **Transaction Integrator** (TI) node to control the operating mode of the TI Manager console.

All computers that have TI installed are able to share the same physical SQL Server configuration database, and TI Manager allows you to edit the configuration of any computer registered in that database from any TI-enabled computer. In HIP, an administrator could edit database entries such as the host environments, objects, or views that are shared among computers. In WIP, an administrator can only edit local WIP settings.

Although multiple system administrators using the TI Manager console can concurrently view, start, or stop any application and listener on any TI-enabled computer registered in the configuration database, only one authorized administrator at a time can create or edit the configuration of the TI database.

As long as the TI Manager is not locked, the administrator can create or edit the following items on any TI-enabled computer:

- Applications
- Listeners on an application
- Local environments
- Host environments
- Security policies
- HIP objects
- Object views

The administrator can also create or edit remote environments and WIP objects on the local computer, and can start, stop, or refresh applications and listeners.

See Also

Reference

Host-Initiated Processing Node
Computers Node
Computer Node
Application
Listener

Host-Initiated Processing Node

Use the **Host-Initiated Processing** node to view the major elements used in host-initiated processing (HIP) environment. The major HIP elements are:

- Computers that have the HIP run-time environment installed. The run-time environment is responsible for accepting incoming requests, instantiating a Microsoft Windows® server object, and returning a reply to the host program that initiated the request.
- Local environments (LEs) that a HIP runtime uses to listen for incoming requests. The LE definitions contain network-transport specific endpoint identification.
- Host environments (HEs) that represent the non-Windows host computers or host environments that deliver requests to a HIP run-time environment. The HE is used by the HIP runtime to define the code page and the data conversion object to be used by the HIP runtime when communicating with the host system identified by the HE.
- Security policies that define how Windows security credentials are established before the server object is invoked.
- Objects that represent the metadata for the server objects that were created through Microsoft Visual Studio®.
- Views define the resolution criteria used by TI when directing the call made from a host environment to a server object. Views restrict access to object methods based on the host environment calling the object and the local environment receiving the call.

Double-click the **Host-Initiated Processing** node to expand the node. The right pane displays the following information about the node:

- **Name**. The name of the major HIP elements.

Right-click the **Host-Initiated Processing** node to display the following five options:

- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of elements as a separate text or Unicode text file.
- **Properties**. Displays the **Host-Initiated Processing Properties** dialog box and one tabbed property page:
    - **Database**. Displays the name of the database.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you

can take.

See Also

Reference

Computers Node
Local Environments Node
Host Environments Node
Security Policies Node
Objects Node (HIP)

Computers Node

Use the **Computers** node to view all the computers that have Transaction Integrator (TI) installed on them and are referenced in the same TI Manager configuration database (MSHIS60_HIP).

Note
If the computer you are looking for does not appear in this list, the computer might be registered in a different configuration database. After you have installed TI, run the **Microsoft Host Integration Server Configuration Wizard** and configure the new computer to use the same Transaction Integrator configuration database as the other computers. You can select the sam e configuration database by changing the **Server** and **Database** settings on the **Database Configurations** wizard page.

Double-click the **Computers** node to expand the node. The right pane displays the following information about the node:

- **Computer**. Displays the name of the HIP-enabled computer.

Right-click the **Computers** node to display the following four options:

- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of computers as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Computer Node

Computer Node

Use the **Computer** node to view and manage the applications running on the computer that is selected in the left pane. An application represents the execution environment for Windows server objects that are initiated by, or driven from, requests from the host computer.

Double-click the **Computer** node to expand the **Computer** node. The right pane displays the following information about the application:

- **Application**. Displays the name of the application.
- **PID**. Displays the process ID of the HIPService. **0** indicates that HIPService is not running.
- **Thread Count**. Displays the minimum number of worker threads the runtime always has active.
- **Maximum Queue Depth**. Displays the maximum number of requests that are stored in the queue waiting to be processed before new requests are rejected.
- **Comment**. Displays additional information about the application.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

   The columns should now be in their original order.

Right-click the **Computer** node to display the following five options:

- **New**.Displays the following menu items:
  - **Empty Application**. Launches the **New Application** dialog box, which allows you to define an application that does not have executable objects associated with it.
  - **Configured Application**. Launches the **New Application Deployment Wizard**, which walks you through the steps of creating the application, local environment, host environment, and objects.
- **View**. Displays the following menu items:
  - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
  - **Large Icons**. Displays large icons for items in the right pane.
  - **Small Icons**. Displays small icons for items in the right pane.

- **List**. Displays only the small icons and the names of the nodes in the right pane.
- **Detail**. Displays the small icons and the properties of the nodes in the right pane.
- **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of applications as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Application

Application

Use the ***application*** **[*status*]** node to start or stop an application or to manage the listeners for an application. An application represents the execution environment for Windows server objects that are initiated by, or driven from, requests from the host computer. An application can host more than one server object and can have more than one listening endpoint associated with it.

The term shown in brackets at the end of the node indicates status of the application:

- **Stopped**. The execution environment and Windows Server 2003 service for the application and its objects are inactive.
- **Incomplete**. The application is missing one or more objects and, therefore, is not available.
- **Starting**. TI Manager is starting the execution environment and Windows Server 2003 service for the application and its objects.
- **Active**. The execution environment for the application is running and the listeners and local environments are active.

Double-click the ***applications*** **[*status*]** node to expand the node. The right pane displays the following information about the listeners:

- **Listener**.The name of the listener.
- **Type**. The type of network (either TCP/IP or SNA).
- **Listening Address**. For a TCP/IP network, the local host. For an SNA network, the local LU defined in the local environment.
- **Endpoints**. The endpoints used to communicate with the host.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

Note
If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Application** node to view the following 12 options:

- **Tracing**. Launches the ***application*** **Trace Options** dialog box. You can select one or more of the following categories for tracing and select one or more options within a category:

    - **General**. Provides high-level information about the end-to-end application processing.
    - **Transport**. Provides detailed connection information about the activities on the SNA or TCP network.
    - **Convert**. Provides detailed information about the data conversion between COM and .NET data on the servers and COBOL or Report Program Generator (RPG) data on the host.
    - **Read Lib**. Provides information about the contents of the type library; used to augment the information in the **Convert** trace.
    - **Flow Control Proxy**. Provides detailed information about object instantiation, state transitions, and method invocation.
- **Reload Definitions**. Performs a dynamic refresh of the application listener's endpoints and resolution tables. An application gets a snapshot of the configuration database at start-up. Any subsequent changes to the configuration do not affect the application until the definitions are reloaded. When you select **Reload Definitions**, the application re-reads all configuration information from the configuration database and applies it immediately. All new requests are processed using the updated resolution information. Working requests that were in progress at the time **Reload Definitions** was clicked are not affected by the update. The resolution information remains unchanged until the work request is completely processed.
- **Start**. Starts the host-initiated processing (HIP) environment for the selected application. Any listeners available for auto-start, as defined on the application property page of a listener that is associated with the application, are started when the application is started. After the application is started, **Start** is disabled and **Stop** is enabled.
- **Stop**. Stops the HIP environment for the selected application. All listeners are stopped when the application is stopped. All pending work is halted immediately and all listening activity is terminated immediately. After the application is stopped, **Stop** is disabled and **Start** is enabled.
- **New**. Displays a list of the following menu items:
    - **Listener**. Launches the **Local Environment** dialog box for you to define a new listening point.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Delete**. Deletes the application from the computer. You can delete an application if it is stopped. The deleted item is removed from the specific computer it was defined on and from the configuration database.
- **Rename**. Renames the selected application. The new name is reflected across all elements of the HIP console.
- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of applications as a separate text or Unicode text file.
- **Properties**. Displays the ***application*** **Properties** dialog box and three tabbed property pages:
    - **General**
    - **Advanced**
    - **.NET assembly path**

  Use the property pages to view or change the properties of the application.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

General Tab (Application Properties)

Advanced Tab (Application Properties)

.NET Assembly Path Tab (Application Properties)

Listener

Use the ***listener* [*status*]** node to start or stop a listener and to determine what views are associated with the listener.

The term shown in brackets at the end of the node indicates status of the listener:

- **Stopped**. The execution environment and Windows Server 2003 service for the listener and its views are inactive.
- **Incomplete**. The listener is missing one or more views or determinants and, therefore, is not available.
- **Unavailable**. The listener was defined after the application was started.
- **Starting**. TI Manager is starting the execution environment and Windows Server 2003 service for the application and its objects.
- **Active**. The execution environment for the application is running and the listeners and local environments are active.

Double-click the ***listener* [*status*]** node to expand the node. The right pane displays the following information about the listener:

- **View**. The name of the view.
- **LE Name**. The name of the local environment.
- **Method Count**. The number of methods.
- **HE Count**. The number of host environments.
- **Comment**. Additional information about the listener.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

Note
If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click on **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the ***listener* [*status*]** node to display the following 10 options:

- **Start**. Starts the host-initiated processing (HIP) service listening on all endpoints defined for the local environment. The HIP application is then ready to receive incoming host requests and to execute methods in accordance with the method

resolution criteria. After the listener is started, **Start** is disabled and **Stop** is enabled. After the listener is started, **[started]** appears to the right of the local environment name in the left pane.

- **Stop**.Stops the HIP service listening on all endpoints defined for the local environment. The HIP application no longer accepts, rejects, services, or queues incoming host requests. After the listener is stopped, **Stop** is disabled and **Start** is enabled. After the listener is stopped, **[stopped]** appears to the right of the local environment name in the left pane.

- **Pause**. Temporarily suspends the HIP service listening on all endpoints defined for the local environment. The application continues to execute work in progress and work items queued for execution, but it no longer accepts or queues incoming requests. A number of requests are queued internally by the low-level network transport components (for example, for TCP, Windows XP stores up to 5 requests while Microsoft Windows Server™ 2003 stores up to 200 requests by default). After the listener is resumed, the pending requests are delivered to the listener. Depending on the number of the processing queue entries available, some of these requests might be finally rejected by the listener. After the listener is paused, **Pause** is disabled, **Resume** is enabled, and **[paused]** appears to the right of the local environment name in the left pane.

- **Resume**. Starts the HIP service listening again on all endpoints defined for the local environment. The HIP application is then ready to receive incoming host requests and to execute methods as defined by the method resolution criteria. After the listener resumes, **Resume** is disabled, **Pause** is enabled, and **[started]** appears to the right of the local environment name in the left pane.

- **View**. Displays the following menu items:

    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.

    - **Large Icons**. Displays large icons for items in the right pane.

    - **Small Icons**. Displays small icons for items in the right pane.

    - **List**. Displays only the small icons and the names of the nodes in the right pane.

    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.

    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Delete**. Deletes the application from the computer. You can delete an application if it is stopped. The deleted item is removed from the specific computer it was defined on and from the administrative data store.

- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of views as a separate text or Unicode text file.

- **Properties**. Displays the ***listener*** **Properties** dialog box and three tabbed property pages:

    - **General**

    - **Endpoints**

    - **Application**

Note
When viewed from this node, the general and endpoint property pages are read-only. If you want to change a general or end point property, right-click the specific ***local environment*** **Node** under the **Local Environments Node**, and then left-click **P roperties** on the shortcut menu.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

View Node (listener)

General Tab (Listener Properties)

Endpoints Tab (TCP/IP Listener Properties)

Endpoints Tab (SNA Listener Properties)

Application Tab (Listener Properties)

View Node (listener)

Use the ***view*** node to determine what methods and host environments are associated with the view.

Double-click the ***view***node to expand the node. The right pane displays either the host environments (HE) associated with the object view or the methods on the view, depending on the choice you select on the shortcut menu.

Right-click the ***view*** node to display the following six options:

- **List**.Displays the following menu items:

    - **HE Associations**.The host environments associated with the view on the application.

    - **Methods**.The methods associated with the view.

    A view can have one or more host environments and one or more methods associated with it. To accommodate listing all the properties of a view, this shortcut menu item allows you to select the properties to be visible in the list view.

- **View**. Displays the following menu items:

    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.

    - **Large Icons**. Displays large icons for items in the right pane.

    - **Small Icons**. Displays small icons for items in the right pane.

    - **List**. Displays only the small icons and the names of the nodes in the right pane.

    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.

    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of views as a separate text or Unicode text file.

- **Properties**. Displays the ***view*** **Properties** dialog box and three tabbed property pages:

    - **General**

    - **Host environments**

    - **Methods**

Note
When viewed from this node, the property pages are read-only. If you want to change a property, right-click the ***view*** **Node** under the ***object*** **Node**, and then left-click **Properties** on the shortcut menu. The properties viewed on those pages can be c hanged.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Host Environment Associations Listing
Methods Listing
General Tab (View Properties)
Host Environments Tab (View Properties)
Methods Tab (View Properties)
View Node (object)

Local Environments Node

Use the **Local Environments** node to manage local environments. A local environment defines how a remote environment contacts the host-initiated processing (HIP) runtime.

Double-click the **Local Environments** node to expand the node. The right pane displays the following information about the local environments:

- **Local Environment**. The name of the local environment.
- **Type**. The type of network used to communicate with the host environment (either TCP/IP or SNA).
- **Transport Class**. The class of the transport object.
- **Endpoint Manager**. The IP address of the local host (for a TCP/IP network); the name of the LU alias (for an SNA network).
- **Endpoints**. The endpoints used to communicate with the host.
- **Comment**. Additional information about the local environment.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

Note
If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click on **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Local Environments** node to view the following five options:

- **New**.Displays the following menu items:
    - **Local environment**. Launches the **New Local Environment Wizard**.
- **View**. Displays the following menu item:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.

- **Small Icons**. Displays small icons for items in the right pane.
- **List**. Displays only the small icons and the names of the nodes in the right pane.
- **Detail**. Displays the small icons and the properties of the nodes in the right pane.
- **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of local environments as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

New Local Environment Wizard

Local Environment Node

Local Environment Node

Use the ***local environment*** node to view and manage the endpoints provided by the local environment.

Double-click the ***local environment***node to expand the node. The right pane displays the following information about the local environment selected in the left pane:

- **Endpoint**. The endpoints used to communicate with the host.

Right-click the ***local environment*** node to display the following five options:

- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of local environments as a separate text or Unicode text file.
- **Properties**. Displays the ***view*** **Properties** dialog box and three tabbed property pages:
    - **General**
    - **Endpoints**
    - **Application**

  Use the property pages to view or change the properties of the local environment.

Note
You can also view the ***local environment*** **Properties** dialog from within the ***application*** node. However, all LE properties ar e read-only when viewed from an application. The **Application** tab appears on the property page when the dialog box is vie wed from an application.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also
Reference
General Tab (Local Environment Properties)
Endpoints Tab (TCP/IP Local Environment Properties)
Endpoints Tab (SNA Local Environment Properties)

Host Environments Node

Use the **Host Environments** node to manage host environments. The host environment defines which host environments can contact the HIP runtime.

Double-click the **Host Environments** node to expand the node. The right pane displays the following information about the node:

- **Host Environment**.The names of the host environments.
- **Type**. The type of network used to communicate with the host environment (either TCP/IP or SNA).
- **Data Conversion**. The data conversion routine used.
- **Send Time-out**. The number of seconds the servicing HIP application waits before it terminates the request if the expected data is not received.
- **Receive Time-out**. The number of seconds the servicing HIP application waits before it terminates the receive function if the expected data is not received.
- **Code Page**. The number of the code page used to transform the incoming and outgoing data to a form that can be used by the host application program to represent the character data.
- **Remote Endpoint Manager**. The IP address of the host or the host name registered in the DNS (for a TCP/IP network); the LU name associated with the host system (for an SNA network).
- **Comment**. Additional information about the host environment.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click on **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Host Environments** node to view the following five options:

- **New**.Displays the following menu item:
    - **Host environment**. Launches the **New Host Environment Wizard**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display

each property as a separate column in the right pane.

- **Large Icons**. Displays large icons for items in the right pane.
- **Small Icons**. Displays small icons for items in the right pane.
- **List**. Displays only the small icons and the names of the nodes in the right pane.
- **Detail**. Displays the small icons and the properties of the nodes in the right pane.
- **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of host environments as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

New Host Environment Wizard

Host Environment Node

Use the ***Host Environment*** node to manage the properties of a host environment.

Right-click the ***Host Environment*** node to view the following three options:

- **Refresh**. Redraws the screen to show any updates.
- **Properties**. Displays the ***host environment*** **Properties** dialog box and four tabbed property pages:
    - **General**
    - **Network**
    - **Conversion**
    - **Default**

  Use these property pages to view or change the properties of the host environment.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

General Tab (Host Environment Properties)
Network Tab (Host Environment Properties)
Conversion Tab (Host Environment Properties)
Default Tab (Host Environment Properties)

Security Policies Node

Use the **Security Policies** node to view and manage security policies. Security policies define how Windows security credentials are established before the server object is run. The security credentials can be based on the user IDs and passwords delivered to HIP by either the client application program or ENTSSO.

Double-click the **Security Policies** node to expand the node. The right pane displays the following information about the node:

- **Security Policy**. The name of the security policy.
- **Credentials Source**.Indicates whether the credentials used are host-based or application-based.
- **Group Application**.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click on **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Security Policies** node to display the following five options:

- **New**.Displays the following menu item:
    - **Security Policy**. Launches the **Security Policy Wizard**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of security policies as a separate text or Unicode text file.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

New Security Policy Wizard

Security Policy Node

Security Policy Node

Use the ***security policy*** node to manage the security policy.

Double-click the ***security policy***node to expand the node. The right pane displays the following information about the security policy selected in the left pane:

- **Affiliate application**. The name of the SSO affiliate application to be queried for access to the Windows credentials needed to execute methods on the server object.

Right-click the ***security policy***node to display the following five options:

- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of security policies as a separate text or Unicode text file.
- **Properties**. Displays the ***security policy*** **Properties** dialog box and three tabbed property pages:
    - **General**
    - **Credentials Source**
    - **Mapping**

    Use the property pages to view or change the properties of the *security policy*.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also
Reference
General Tab (Security Policy Properties)
Credentials Source Tab (Security Policy Properties)

Objects Node (HIP)

Use the **Objects** node to view and manage COM and .NET Framework objects accessible from a host. Objects contain the metadata definitions for the server objects created as TI Projects in Visual Studio.

Double-click the **Objects** node to expand the node. The right pane displays the following information about the node:

- **Object**. The name of the object entry. The default name is the ProgID (for COM) or Namespace.Interface (for .NET), but you can change the name on the *object* context menu.
- **Type**. The type of object.
- **Component**. The programming identifier of the COM object or the interface name of the .Net object.
- **Method Count**. The number of methods on the object.
- **Comment**. Additional information about the object.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click on **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Objects** node to display the following five options:

- **New**.Displays the following menu item:
    - **Object**. Launches the **HIP Object Wizard**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.

    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of objects as a separate text or Unicode text file.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Object Wizard (for HIP)

Object Node (HIP)

Object Node (HIP)

Use the ***object*** node to create and manage object views. An object view is a representation of the server object itself. A view can include one or more methods of an object, and not all methods have to be defined in a view. You can define one or more views for an object, which enables you to restrict the methods of an object to specific endpoints and host environment pairs.

Double-click the ***object***node to expand the node. The right pane displays the following information about the view selected in the left pane:

- **View**. The name of the view.
- **LE Name**. The name of the local environment.
- **Security Policy**. The name of the security policy.
- **Method Count**. The number of methods.
- **HE Count**. The number of host environments.
- **Comment**. Additional information about the view.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the ***object***node to display the following six options:

- **New**. Displays the following menu item:
    - **View**. Launches the **New Object View Wizard**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.

    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of objects as a separate text or Unicode text file.
- **Properties**. Displays the ***object*** **Properties** dialog box and two tabbed property pages:
    - **General**
    - **Methods**

  Use these property pages to view or change the properties of the object.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

General Tab (Object Properties)
Methods Tab (Object Properties)
New Object View Wizard

View Node (object)

Use the ***view*** node to restrict access to object methods based on the host environment calling the object and the local environment receiving the call. You also use the ***view*** node to define the resolution criteria used to direct the call made from a host environment to a server object.

Double-click the ***view*** node to expand the node. The right pane displays either the host environments (HE) associated with the object view or the methods on the object view, depending upon the choice you select on the shortcut menu.

Right-click the ***view*** node to display the following seven options:

- **List**.An object view can have one or more host environments and one or more methods associated with it. To accommodate a list of all the properties of an object view, this shortcut menu item enables you to select which properties are to be viewed in the list view.

    - **HE Associations**.Displays the host environments associated with the object view on the application.

    - **Methods**.Displays the methods associated with the object view.

- **Delete**. Deletes the object view.

- **Rename**. Renames the selected object view. The new name is reflected across all elements of the HIP console

- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of views as a separate text or Unicode text file.

- **Properties**. Displays the ***view*** **Properties** dialog box and three tabbed property pages:

    - **General**

    - **Host environments**

    - **Methods**

    Use these property pages to view or change the properties of the object view.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Host Environment Associations Listing
Methods Listing
General Tab (View Properties)
Host Environments Tab (View Properties)
Methods Tab (View Properties)

Host Environment Associations Listing

Use the host environment associations listing to view the host environments associated with the view.

This listing displays the following information about the host environment:

- **Host Environment**.The name of the host environment.
- **Type**. The type of network used to communicate with the host environment (either TCP/IP or SNA).
- **Remote Endpoint Manager**.The IP address of the host or the host name registered in the DNS (for a TCP/IP network); the LU name associated with the host system (for an SNA network).
- **Comment**.Additional information about the host environment.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the host environmentname to display the following four options:

- **Delete**. Deletes the host environment association (not the host environment itself). You can delete the host environment association anytime.
- **Refresh**. Redraws the screen to show any updates.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

General Tab (Host Environment Properties)
View Node (object)
Methods Listing

Methods Listing

Use the methodslistingto view the properties of the methods on the object view.

This listing displays the following information about the method:

- **Method**. The name of the method.
- **Endpoint**. The endpoint used to communicate with the host.
- **Resolution Type**. The type of conflict to be conducted in case of a conflict.
- **Msg Handler**.
- **Resolution String**.
- **Position**. The starting point in the data stream where the TI runtime looks for the data defined in the resolution data field to determine what method is to be executed.
- **Comment**. Additional information about the view.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** c ommand on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manag er. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column hea dings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the methodname to view the following two options:

- **Refresh** Redraws the screen to show any updates.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also
Reference
Host Environment Associations Listing
View Node (object)

Windows-Initiated Processing Node

Use the **Windows-Initiated Processing** node to view the major elements used in Windows-initiated processing (WIP) environment. The major WIP elements are:

- Remote environments
- Objects

Double-click the **Windows-Initiated Processing** node to expand the node. The right pane displays the following information about the node:

- **Name**. The name of the major WIP elements.

Right-click the **Windows-Initiated Processing** node to view the following five options:

- **Tracing**. Launches the **Windows-Initiated Processing Trace Options** dialog box. You can select one or more of the following categories for tracing and select one or more options within a category:
    - **General**. Provides high-level information about the end-to-end global processing.
    - **Transport**. Provides detailed connection information about the activities on the SNA or TCP network.
    - **Convert**. Provides detailed information about the data conversion between COM and .NET data on the servers and COBOL or RPG data on the host.
    - **Read Lib**. Provides information about the contents of the type library; used to augment the information in the **Convert** trace.
    - **Proxy**. Provides detailed information about object instantiation, state transitions, and method invocation.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of elements as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

In This Section

- Remote Environments Node
- Remote Environment Node
- Objects Node (WIP)
- Object Node (WIP)

Remote Environments Node

Use the **Remote Environments** node to manager remote environments. A remote environment defines the characteristics of the non-Windows host environment that receives requests from Windows-initiated processing (WIP) components.

Double-click the **Remote Environments** node to expand the node. The right pane displays the following information about the node:

- **Remote Environment**. The names of the remote environment.
- **Type**. The type of remote environment.
- **Comment**. Additional information about the remote environment.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Remote Environments** node to display the following five options:

- **New**. Displays the following menu items:
    - **Remote Environment**. Launches the **New Remote Environment Wizard**.
    - **Capture RE**.Launches the **New Remote Environment Wizard** to create a remote environment type that captures responses and saves those responses in a recording file. After being saved, the responses can be played back using the playback remote environment.
    - **Playback RE**.Launches the **New Remote Environment Wizard** to create a remote environment type that plays back responses stored in a recording file.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.

    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of remote environments as a separate text or Unicode text file.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

New Remote Environment Wizard

Remote Environment Node

Remote Environment Node

Use the ***remote environment*** node to view and manage a remote environment.

Double-click the ***remote environment*** node to expand the node. The right pane displays the following information about the remote environment selected in the left pane:

- **Component**. The name of the component.
- **Type**.
- **Comment**. Additional information about the remote environment.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the ***remote environment*** node to display the following nine options:

- **Activate**.
- **Deactivate**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.
    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.
- **Delete**. Deletes the remote environments selected in the ***remote environments*** node. The deleted item is removed from the ***remote environments*** node and from the administrative data store. If an RE is deleted while it still has objects assigned to it, the objects become unassigned.

- **Rename**. Renames the selected remote environment. The new name is reflected across all the elements of the WIP console.
- **Refresh**. Redraws the screen to show any updates.
- **Export List**. Allows you to save the list of remote environments as a separate text or Unicode text file.
- **Properties**. Displays the ***remote environment*** **Properties** dialog box and several tabbed property pages:
    - **General**
    - **TCP/IP**
    - **LU 6.2**
    - **MQ Series**
    - **Target**
    - **Recording**
    - **Locale**
    - **Security**
    - **CICS**
    - **IMS**

  Use these property pages to view or change the properties of the remote environment.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

General Tab (Remote Environment Properties)
TCP/IP Tab (Remote Environment Properties)
LU6.2 Tab (Remote Environment Properties)
Target Tab (Remote Environment Properties)
Recording Tab (Remote Environment Properties)
Locale Tab (Remote Environment Properties)
Security Tab (Remote Environment Properties)

Objects Node (WIP)

Use the **Objects** node to view and manage the objects.

Double-click the **Objects** node to expand the node. The right pane displays the following information about the node:

- **Component**. The name of the component.
- **Type**. The type of component.
- **RE Type**. The type of remote environment.
- **Application Host**. The name of the COM+ application or IIS virtual directory the component is assigned to.
- **Remote Environment**. The name of the remote environment.
- **Comment**. Additional information about the component.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the **Objects** node to display the following five options:

- **New**. Displays the following menu item:
    - **Object**. Launches the WIP **Object Wizard**.
- **View**. Displays the following menu items:
    - **Add/Remove Columns**. Allows you to choose the properties that are displayed in the list view. You can display each property as a separate column in the right pane.
    - **Large Icons**. Displays large icons for items in the right pane.
    - **Small Icons**. Displays small icons for items in the right pane.
    - **List**. Displays only the small icons and the names of the nodes in the right pane.
    - **Detail**. Displays the small icons and the properties of the nodes in the right pane.

    - **Customize**. Allows you to change the options to show or hide items displayed in the right pane.

- **Refresh**. Redraws the screen to show any updates.

- **Export List**. Allows you to save the list of objects as a separate text or Unicode text file.

- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also

Reference

Object Wizard (for WIP)

Object Node (WIP)

Object Node (WIP)

Use the ***object*** node to manage an object.

Double-click the ***object*** node to expand the node. The right pane displays the following information about the node:

- **Method**. The name of the methods on the object.
- **Transaction/Program**.

Important
If you view the list in the right pane of TI Manager and change the order of the columns using the **Add/Remove Columns** command on the shortcut menu, the order of the column headings might not be correct after you close and re-open TI Manager. The next time you open TI Manager, the values in the columns are displayed in the new order you set, but the column headings are in the their default order.

If you observe that the column headings are no longer in the order you set:

1. Right-click the node, point to **View**, and then click **Add/Remove Columns**.
2. On the **Add/Remove Columns** dialog box, click **Restore Defaults**.
3. Click a different node, then click the node you were just working on.

Note
The columns should now be in their original order.

Right-click the ***object*** node to display the following four options:

- **Delete**. Deletes the selected object.
- **Refresh**. Redraws the screen to show any updates
- **Properties**. Displays the ***object*** **Properties** dialog box and two tabbed property pages:
    - **General**
    - **Hosting**

  Use these property pages to view or change the properties of the *object*.
- **Help**. Displays a Help topic (this topic) that explains the items that appear in the TI Manager console and the actions you can take.

See Also
Reference
General Tab (Object Properties)
Hosting Tab (COM Object Properties)

TI Manager Wizards and Dialog Boxes

Transaction Integrator (TI) Manager provides wizards and special dialog boxes to help you complete various tasks.

The following wizards help you with host-initiated processing (HIP).

New Application Dialog Box

Credentials for Service <application name> Dialog Box

New Application Deployment Wizard

Define Implementation Characteristics for the .NET Object Wizard Page

New Local Environment Wizard

New Host Environment Wizard

New Security Policy Wizard

Object Wizard (for HIP)

New Object View Wizard

Local Environment Dialog Box

Reload TIMs Wizard

The following wizards help you with Windows-initiated processing (WIP):

New Remote Environment Wizard

Object Wizard (for WIP)

New Application Dialog Box

Use the **New Application** dialog box to define an application that does not have executable objects associated with it.

Use this	To do this
Computer name	View the name of the computer where the application runs.
Application	Type the name of the application. The name can be a maximum of 25 alpha-numeric characters. The first character in the name must be a letter, and the name cannot contain any embedded spaces.
Comment	Type a comment that provides additional information about the use of the application in the HIP environment. The comment can be a maximum of 259 alpha-numeric characters.

See Also

Other Resources

TI Manager Wizards and Dialog Boxes

Credentials for Service <application name> Dialog Box

Use the **Credentials for Service <application name>** dialog box to identify the security credentials used by the application.

Use this	To do this
User ID	Type the user ID of the security credentials.
Password	Type the password of the security credentials.

See Also

Other Resources

TI Manager Wizards and Dialog Boxes

New Application Deployment Wizard

The **New Application Deployment Wizard** helps you add a new application by guiding you through the definitions of the application, local environment, host environment, and objects. The wizard can be used to define additional local environments, host environments, or objects and link them to existing definitions. You can access the Wizard from the shortcut menu on the ***computer*** node in the TI Manager console tree.

In This Section

Welcome to the New Application Deployment Wizard Page

Use the **Welcome to the New Application Deployment Wizard** page to view the definition of an application deployment and to control whether the welcome page is displayed the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Create an Application Wizard Page

Create an Application Wizard Page

Use the **Create an Application** wizard page to identify the application to be deployed and provide an additional description. Type the name of the new application.

Use this	To do this
Application	Type the name for the application. The name can be a maximum of 25 alpha-numeric characters. The first character in the name must be a letter, and the name cannot contain any embedded spaces. The name cannot be the same as that of an existing application.
Comment	Type any additional information about the application. The comment can be a maximum of 259 Unicode characters.

See Also

Reference

Configure a New Local Environment Wizard Page

Configure a New Local Environment Wizard Page

Use the **Configure a New Local Environment** wizard page to identify the basic characteristics of the new local environment. Select an existing local environment from the dropdown list or type the name of the new local environment. To define its characteristics, select a specific network type and transport class.

Use this	To do this
Local environment	Type the name for the local environment. The name can be a maximum of 259 alpha-numeric characters. The name cannot be the same as that of an existing local environment.
Network type	Select the type of network used to communicate with the host.
Transport class	Select the class of the transport object. The list changes according to the selected **Network type**.
Endpoint manager	For a TCP/IP network, this is always set to the local host and disabled. For an SNA network, type the name of the LU alias. In either case, the name can be a maximum of 259 Unicode characters.

See Also

Reference

Configure Local Environment Endpoints (TCP/IP) Wizard Page

Configure Local Environment Endpoints (SNA) Wizard Page

Configure Local Environment Endpoints (TCP/IP) Wizard Page

Use the **Configure Local Environment Endpoints** (TCP/IP)wizard page to define the endpoints used to communicate with the host. To define the TCP/IP ports or service names for the local environment, type a port number or name, and then click **Add**.

Use this	To do this
Local environment	View the name of the local environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
New port or service name	Type the port number or service name of the local environment. The port number can be a maximum of 5 numeric characters and must be greater than 0 and less than 32768. The service name can be a maximum of 259 alpha-numeric characters. The format of the name must conform to the WinSock Service Name convention. The port number or name cannot be the same as that of an existing number or name in the **Defined ports and service names**.
Add	Click to validate the port number or service name you typed. If the number or name is valid, it appears in **Defined ports and service names**.
Defined ports and service names	View the existing valid ports and services names for the local environment.
Remove	Click to remove a selected port number or service name from the **Defined ports or service names** available for use in the local environment.

See Also
Reference
Configure a New Host Environment Wizard Page

Configure Local Environment Endpoints (SNA) Wizard Page

Use the **Configure Local Environment Endpoints** (SNA)wizard page to define the endpoints used to communicate with the host. Type a transaction program name or mirror transaction ID, and then click **Add**.

Use this	To do this
Local environment	View the name of the local environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
New transaction program name	Type the transaction program (TP) name or the link mirror transaction ID of the local environment. The name or ID can be a maximum of 64 alphabetic characters. The format of the name must conform to the IBM SNA Transaction Program Names convention. The name or ID cannot be the same as that of an existing name or ID in the **Defined transaction program names**.
Add	Click to validate the transaction program (TP) name or the link mirror transaction ID you typed. If the TP name or the link mirror transaction ID is valid, it appears in **Defined transaction program names**.
Defined transaction program names	View the existing TP names or the link mirror transaction IDs for the local environment.
Remove	Click to remove one or more TP names or link mirror transaction IDs selected from the **Defined transaction program names** available for use in the local environment.

See Also
Reference
Configure a New Host Environment Wizard Page

Configure a New Host Environment Wizard Page

Use the **Configure a New Host Environment** wizard page to describe the basic host characteristics that are concerned with data formats. Type the name of the host environment, and then type or select the attributes of the host.

Use this	To do this
Host environment	Type the name for the host environment. The name can be a maximum of 259 alpha-numeric characters. The name cannot be the same as that of an existing host environment. Caution If the host environment is located on the same computer as the HIP application, and the client program uses either host name **localhost** or IP address **127.0.0.1** to connect to the HIP application, the host environment should also use **localhost** or **127.0.0.1** instead of the real computer name or IP address. Failure to set the names or IP addresses identical for the host and client prevents the HIP runtime from delivering calls between the two, and causes the application to fail.
Data conversion	Select the data conversion routine used.
Host code page	Select the code page used by the host. Each available code page is listed by its symbolic value.
Network type	Select the type of network used to communicate with the host. The choices are: • TCP/IP (default) • SNA
Remote endpoint manager	The IP address of the host or the host name registered in the DNS (for a TCP/IP network); the LU name associated with the host system (for an SNA network). In either case, the name can be a maximum of 259 alpha-numeric characters.

See Also

Reference

Configure Host Environment Default Method Resolution Wizard Page

Configure Host Environment Default Method Resolution Wizard Page

Use the **Configure Host Environment Default Method Resolution** wizard page to identify the default characteristics of the typical interaction of the applications programs on the host with the application programs on the server. Select the default method resolution criteria for the host.

Use this	To do this
Host environment	View the name of the host environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
Default method resolution criteria	View the definition of the default behavior for setting up the method resolution criteria for a new object view.
Type	Select the type of resolution to be conducted: **Endpoint:** The endpoint resolution type is the simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables the **Object**, **Input format** and **Output format** boxes and clears their contents. • **Transaction Request Message (TRM):** The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant, MSLink model. It represents a double exchange sequence. The first exchange represents the transaction request; the second exchange represents the request and reply data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Transaction Request Message (TRM)** enables the **Object**, **Input format**, and **Output format** boxes. • **Enhanced Listener Message (ELM)**: • **Data:** Similar to **Endpoint** in that it adds the ability to identify a string in the data stream directed to a port that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Data** disables the **Object**, **Input format**, and **Output format** boxes and clears their contents. • **Link to Program Name:** Select **Link to Program Name** to resolve all the methods of a new object view using the data information specified when the object view was created. When a new view is created and associated with the HE, the method resolves to the Link Tran ID field in the default HE resolution. Selecting **Link to Program Name** disables the **Object**, **Input format**, and **Output format** boxes and clears their contents.
Object	Select the object. The list of objects is populated from the Help string in the interface definition of objects defined as TRM resolution handlers. Microsoft provides three TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS**

Input format	Select the format of the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is determined by the user-defined type definitions in the MicrosoftTRMDef.tim file. The **Input format** box is blank by default, but three input formats are defined that match the names of the resolution handler **Object**: • **TRMINMSLink** • **TRMINMSCCS** • **TRMINIBMCCS**
Output format	Select the format of the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream that is returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is determined by the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is blank by default, but three output formats are defined that match the names of the resolution handler **Object**: • **MS Link Reply Format** • **IBM Listener Type 1 Reply Format** • **IBM Listener Type 2 Reply Format**
Link Tran ID	View or select an existing TP name or the link mirror transaction ID for the local environment.

See Also

Reference

Configure a New Security Policy Wizard Page (in the New Application Deployment Wizard) (1)

Configure a New Security Policy Wizard Page (in the New Application Deployment Wizard) (1)

Use the first **Confige a New Security Policy** wizard page to identify the security policy and the source of the credentials. Type the name of the security policy, and then select the source for the credentials.

Use this	To do this
Security policy	Type the name for the security policy. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as the name of an existing security policy. The name for the new security policy is a required field; you will receive an error message if you click **Next** without providing a name. The name is used to associate the security policy with an object view in other wizard pages and dialog boxes.
Credentials source used to invoke the server	Identify the type or source of credentials that host-initiated processing will associate with the thread when the method on the server object is executed.
Host-initiated Single Sign-on	Select this option to use host user ID and password. This option is automatically disabled if Single Sign-on (SSO) is not installed or not available.
Windows credentials of the HIP application	Select this option to use Windows user ID and password specified on the HIP NT application service. This option is automatically selected if SSO is not installed or not available.

See Also

Reference

Configure a New Security Policy Wizard Page (in the New Application Deployment Wizard) (2)

Configure a New Security Policy Wizard Page (in the New Application Deployment Wizard) (2)

Use the second **Configure a New Security Policy** wizard page to identify the source of the host user ID and password and describe how they are translated into Windows-based credentials. This wizard page appears only if **Host-initiated Single Sign-on** (SSO)is selected on the previous **Configure a new security policy** wizard page.

Use this	To do this
Security policy	View the name of the security policy.
Affiliate application	Select the SSO affiliate application to be queried to gain access to the Windows credentials needed to execute methods on the server object. The list displays all the affiliate applications defined in the SSO. The display name of the affiliate application is a concatenation of the affiliate application name and description, separated by a hyphen. The selected application will be added to the security policy. Note The dropdown list and **Next** are disabled if SSO is not installed or is not available.
Single Sign-on mapping use	Specify the credentials to be used when mapping to Single Sign-on.
Default credentials	Select this option to enter the user ID and password used if the request from the host does not contain a user ID and password or the user ID and password are set to spaces or nulls. This option enables the **Group application** and **User** boxes.
Group application	The default user group provides the default user ID and password.
User	Type or select the host user name used in the lookup call to SSO. Note The dropdown list and Next are disabled if SSO is not installed or is not available.

See Also

Reference

Specify or Locate an Object Wizard Page

Specify or Locate an Object Wizard Page

Use the **Specify or Locate an Object** wizard page to identify objects associated with the application. Select an object from the list, or, to create a new object, type the new object name.

Use this	To do this
Object	Select the name of the object. To create a new object, type the new object name. If you select the Transaction Integrat or metadata (TIM) file first, the name is populated with the ProgID or Namespace.Interface defined in the TIM.
Registered TIMs	Type the full path name, including the *.tim extension, to a TIM file, select a recently used file from the list, or click **Browse** and navigate to the file.

See Also

Reference

Configure a New Object View Wizard Page

Configure a New Object View Wizard Page

Use the **Configure a New Object View** wizard page to identify the object view associated with the application. Select the option to add a new object view or the option to use existing object views.

Use this	To do this
Add a new view	Select this option to create a new object view.
Use existing views associated with the local environment *name*	Select this option to add all object views already associated with the specific local environment. This option is disabled if there are no existing views associated with the local environment.

See Also

Reference

Configure the New View Wizard Page

Configure the New View Wizard Page

Use the **Configure the New View** wizard page to select the methods that are included in the object view. Type the name of the view. To specify the methods to be included in the view, select the check box by the method name. Double-click the method name to edit it.

Use this	To do this
View	Type the name of the object view. The name can be a maximum of 259 alpha-numeric characters. The name cannot be the same as that of an existing view.

Object methods	View all the methods that are defined in the object. Each row contains a summary of the resolution information for a single method. Rows in the list cannot be deleted. The columns are:
	• **Include** Select this option to include the method in the object view.
	• **Method** View the name of the method. Double-click the method name displays the **Method Resolution Criteria** dialog.
	• **Endpoints** View one of the endpoints defined in the local environment associated with the view. To change the content of this field, double-click on the **Endpoints** field for a specific method.
	• **Resolution Type** View one of the possible resolution types:
	  • **Endpoint**
	  • **Transaction Request Message (TRM)**
	  • **Enhanced Listener Message (ELM)**
	  • **Data**
	  • **Link to Program**
	To change the content of this field, double-click the **Resolution Type** field for a specific method.
	• **Resolution Data** Identifies the data used to select the method to be executed. To change the content of this field, double-click the **Resolution Data** field for a specific method.
	• **Resolution Position** View the resolution position. If **Data Resolution** is selected, this information is used by the TI runtime to identify the starting point in the data stream in which to look for the data defined in the resolution data field to determine what method is to be executed. To change the content of this field, double-click the **Resolution Position** field for a specific method.

See Also

Reference

Method Resolution Criteria Dialog Box (in the New Application Deployment Wizard)

Method Resolution Criteria Dialog Box (in the New Application Deployment Wizard)

Use the **Method Resolution Criteria** dialog to provide detailed information about a specific method on the object view.

Use this	To do this
Method	View the name of the object's method for which the method resolution criteria are being set.
Endpoint	Select the endpoints on the local environment that are available for the type of resolution specified in **Type**. All endpoints in the LE are listed and available for selection. If an endpoint selected for one resolution type is already used with another resolution type then an error message will pop up when you click **Apply**. The error message explains which object and view has this endpoint used with another resolution type.
Method resolution criteria	View a description of the behavior for the IT runtime when it resolves a request to a method. The sequence of data flow events may require the TI to send and receive various amounts of data prior to obtaining the ability to make the connection between the incoming request and the target method.
Type	Select the type of resolution to be conducted: • **Endpoint** The endpoint resolution type is simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and it represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables **Object Name**, **Input format** and **Output format** and clears their contents. • **Transaction Request Message (TRM)** The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant MSLink model. It represents a double exchange sequence. The first exchange represents the transaction request, and the second exchange represents the request and reply data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Transaction Request Message (TRM)** enables **Object**, **Input format**, and **Output format**. This option is available only for TCP/IP local environment/host environment pairs. • **Enhanced Listener Message (ELM)**: • **Data** Similar to **Endpoint** in that it adds the flexibility to identify a string in the data stream directed to a port that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Data** disables **Object**, **Input format**, and **Output format** and clears their contents. • **Link to Program Name:** Resolves all the methods of a new object view using the data information specified when the object view was created. When a new view is created and associated with the HE, the method resolves to the Link Tran ID field in the default HE resolution. Selecting **Link to Program Name** disables **Object**, **Input format**, and **Output format** and clears their contents. This option is available only for SNA local environment/host environment pairs. Note Only **TRM-MSLink** and **Link to Program Name** are available for Link methods. Methods can be defined as Link in Visual Studio by setting the **Is Link** property to **True**.

Object Name	Select the object. The list of objects is populated from the Help string in the interface definition of objects identified to the resolution handler. Microsoft provides three TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS**
Input format	Select the format of the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is determined by the UDT) definitions in the MicrosoftTRMDef.tim file in the installation TIMLibs directory. The **Input format** box is blank by default, but three input formats are defined that match the names of the resolution handler **Object**: • **MS Link Request Format** • **IBM Listener Type 1 Request Format** • **IBM Listener Type 2 Request Format**
Output format	Select the format of the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream that is returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is determined by the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is blank by default, but three output formats are defined that match the names of the resolution handler **Object**: • **MS Link Reply Format** • **IBM Listener Type 1 Reply Format** • **IBM Listener Type 2 Reply Format**
Link	View the name of the program being linked to.
Program Name	Name of the TP program.
Resolution Data	View a definition of the data used by the IT runtime when it resolves a request to a method.
Data	Type the method resolution criteria for a method on the object view. The criteria can be a maximum of 256 alpha-numeric characters. This control is enabled for resolution types **Transaction Request Message (TRM)** and **Data**. This control is disabled for resolution type **Endpoint**.
Position	Type the resolution position. The position can be a maximum of nine numeric characters. This control is enabled for resolution type **Data** and disabled for resolution types **Transaction Request Message (TRM)** and **Endpoint**.

See Also

Reference

Define Implementation Characteristics for the .NET Object Wizard Page

Define Implementation Characteristics for the .NET Object Wizard Page

Use the **Define Implementation Characteristics for the .NET Object** wizard page to identify a .NET assembly implementing the interface defined in the Transaction Integrator metadata (TIM) file.

Use this	To do this
Defer implementation identification	Select this option to identify the implementing assembly later through the object's **Properties** dialog.
Pick assembly	Select this option to identify the implementing assembly now.
Path	Type the full path name, including the .dll extension, to a .NET file, select a recently used file from the list, or click **Browse** and navigate to the file.
Class	Select the class containing the methods to be called by HIP runtime for the request processing. The list displays classes from the selected assembly that implement the interface defined in the .TIM file. The first class is selected in the list by default.

See Also

Reference

Completing the New Application Deployment Wizard Page

Completing the New Application Deployment Wizard Page

Use the **Completing the Application Deployment Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
New Application Deployment Wizard

New Local Environment Wizard

The **New Local Environment Wizard** helps you define the network transport endpoints for the Windows operating system environment. The wizard collects information about the following:

- Local environment name.
- Network transport type.
- Network transport class.
- Endpoint manager.
- Endpoint identification.

In This Section

Welcome to the New Local Environment Wizard Page

Use the **Welcome to the New Local Environment Wizard** page to view the definition of a local environment and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Configure a New Local Environment Wizard Page

Configure a New Local Environment Wizard Page

Use the **Configure a New Local Environment** wizard page to define the basic characteristics of the new local environment. Type the name of the new local environment. To define its characteristics, select a specific network type and transport class.

Use this	To do this
Local environment	Type the name for the local environment. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as that of an existing local environment. The name for the new local environment is a required field, and you will receive an error message if you click **Next** without providing a name.
Network type	Select the type of network used to communicate with the host. The choices are: • TCP/IP (default) • SNA
Transport class	Select the class of the transport object. The list of choices changes according to the selected **Network type**. The default transport class is TCPTransport Class.
Endpoint manager	Select or type the local LU alias to manage the endpoint. Note The Endpoint manager edit box is set to localhost and disabled if the Network type is set to TCP/IP.

See Also

Reference

Configure Local Environment Endpoints (TCP/IP) Wizard Page

Configure Local Environment Endpoints (SNA) Wizard Page

Configure Local Environment Endpoints (TCP/IP) Wizard Page

Use the **Configure Local Environment Endpoints** (TCP/IP)wizard page to define the endpoints used to communicate with the host. To define the TCP/IP ports or service names for the local environment, type a port number or name, and then click **Add**.

Use this	To do this
Local environment	View the name of the local environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
New port or service name	Type the port number or service name of the local environment. The port number can be a maximum of 5 numeric characters and must be greater than 0 and less than 32768. The service name can be a maximum of 259 alpha-numeric characters. The format of the name must conform to the WinSock Service Name convention. The port number or name cannot be the same as that of an existing number or name in the **Defined ports and service names**.
Add	Click to validate the port number or service name you typed. If the number or name is valid, it appears in **Defined ports and service names**.
Defined ports and service names	View the existing valid ports and services names for the local environment.
Remove	Click to remove a selected port number or service name from the **Defined ports or service names** available for use in the local environment.

See Also

Reference

Completing the New Local Environment Wizard

Configure Local Environment Endpoints (SNA) Wizard Page

Use the **Configure Local Environment Endpoints** (SNA)wizard page to define the endpoints used to communicate with the host. Type a transaction program name or mirror transaction ID, and then click **Add**.

Use this	To do this
Local environment	View the name of the local environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
New transaction program name	Type the transaction program (TP) name or the link mirror transaction ID of the local environment. The name or ID can be a maximum of 64 alphabetic characters. The format of the name must conform to the IBM SNA Transaction Program Names convention. The name or ID cannot be the same as that of an existing name or ID in the **Defined transaction program names**.
Add	Click to validate the transaction program (TP) name or the link mirror transaction ID you typed. If the TP name or the link mirror transaction ID is valid, it appears in **Defined transaction program names**.
Defined transaction program names	View the existing TP names or the link mirror transaction IDs for the local environment.
Remove	Click to remove one or more TP names or link mirror transaction IDs selected from the **Defined transaction program names** available for use in the local environment.

See Also

Reference

Completing the New Local Environment Wizard

Completing the New Local Environment Wizard

Use the **Completing the New Local Environment Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also

Reference

New Local Environment Wizard

New Host Environment Wizard

The **New Host Environment Wizard** helps you define the network and hardware characteristics of the non-Windows host that will be initiating requests to the Windows operating system. The wizard collects information about the following:

- Host environment name
- Host identification
- Network transport type
- Data conversion information
- Default method resolution

In This Section

Welcome to the New Host Environment Wizard

Use the **Welcome to the New Host Environment Wizard** page to view the definition of a host environment and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Configure a New Host Environment Wizard Page

Configure a New Host Environment Wizard Page

Use the **Configure a New Host Environment** wizard page to identify the data format characteristics of the host. Type the name of the host environment, and then type or select the attributes of the host.

Use this	To do this
Host environment	Type the name for your host environment. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as that of an existing host environment. The name for the new host environment is a required field, and you will receive an error message if you click Next without providing a name. ⚠Caution If the host environment is located on the same computer as the HIP application, and the client program uses either host name **localhost** or IP address **127.0.0.1** to connect to the HIP application, the host environment should also use **localhost** or **127.0.0.1** instead of the real computer name or IP address. Failure to set the names or IP addresses identical for the host and client prevents the HIP runtime from delivering calls between the two, and causes the application to fail.
Data conversion	Select the data conversion routine to be used.
Host code page	Select the code page used by the host. Each available code page is listed by its symbolic value.
Network type	Select the type of network used to communicate with the host. The choices are: • TCP/IP (default) • SNA
Remote endpoint manager	The IP address of the host or the host name registered in the DNS (for a TCP/IP network); the LU name associated with the host system (for an SNA network). In either case, the name can be a maximum of 259 alpha-numeric characters.

See Also

Reference

Configure Host Environment Default Method Resolution Wizard Page

Configure Host Environment Default Method Resolution Wizard Page

Use the **Configure Host Environment Default Method Resolution** wizard page to identify the default characteristics of the typical interaction between applications programs on the host and the application programs on the server. Type or select the default method resolution criteria for the host.

Use this	To do this
Host environment	View the name of the host environment. The name was entered on a previous page of the wizard. To change the name, click **Back**.
Default method resolution criteria	View a definition of the default behavior for setting up the method resolution criteria for a new object view.

Type	Select the type of resolution to be conducted: • **Endpoint:** The endpoint resolution type is the simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and represe nts a single synchronous exchange of data. When a new view is created and associated with the HE, the method re solves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables the **Object**, **I nput format**, **Output format**, and **Link Tran ID** boxes and clears their contents. • **Transaction Request Message (TRM):** The TRM resolution type is specific to the TCP/IP transport and to the IB M CICS Concurrent Server model and the Microsoft variant MSLink model. It represents a double exchange seque nce. The first exchange represents the transaction request; the second exchange represents the request and reply data. Select **Transaction Request Message (TRM)** to resolve all the methods of a new object view using the resolutio n handler information in the **Object**, **Input format**, and **Output format** boxes. When a new view is created and a ssociated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selec ting one of the predefined **Transaction Request Objects** disablesthe **Input format** and **Output format** boxes; i t also disables the **Link Tran ID** box and clears its contents. Selecting a custom object enables the **Input format** a nd **Output format** boxes and sets their initial values to the Microsoft supplied TRM handlers items. • **Enhanced Listener Message (ELM)**: A streamlined, application-level protocol exchange sequence that sends to and receives from the host application a single data stream composed of a header followed by the application dat a. • **Data:** Similar to the **Endpoint** type in that it enables you to identify a string in the data stream directed to a port t hat is used to associate the request with a specific method in the view. This model is considered "raw sockets." It r epresents a single synchronous exchange of data. When a new view is created and associated with the HE, the me thod resolves to the first endpoint defined in the associated local environment. Selecting **Data** disables the **Objec t**, **Input format**, **Output format,** and **Link Tran ID** boxes and clears their contents. • **Link to Program Name:** The Link resolution type is specific to the IBM CICS DPL model. The endpoint name is re presentative of the TP name that is the CICS mirror transaction ID. Select **Link to Program Name** to resolve all the methods of a new object view using the data information specifi ed when the object view was created. When a new view is created and associated with the HE, the method resolve s to the Link Tran ID field in the default HE resolution. Selecting **Link** disables the **Object**, **Input format**, and **Out put format** boxes and clears their contents.
Objec t	Select the object. The list of objects is populated from the Help string in the interface definition of objects identified to t he resolution handler. Microsoft provides four TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS** • **Microsoft HIS - Link Handler**

Input format	Select the format for the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is populated from the user-defined type definitions in the MicrosoftTRMDef.tim file. The **Input format** box is disabled for standard TRM handlers and enabled for custom TRM handlers, but three input formats are defined that match the names of the resolution handler **Object**: • **TRMINMSLink** • **TRMINMSCCS** • **TRMINIBMCCS**
Output format	Select the format for the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream to be returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is populated from the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is disabled for standard TRM handlers and enabled for custom TRM handlers, but three output formats are defined that match the names of the resolution handler **Object**: • **TRMOUTMSLink** • **IBM Listener Reply Format (TRMOUTCCS)** • **Link Tran ID:** Select the TP name or Link Mirror Tran ID for the SNA local environment. This control is enabled only when the **Network Type** is set to SNA and the **Resolution Type** is set to Link.
Link Tran ID	View or select an existing TP name or the link mirror transaction ID for the local environment.

See Also
Reference
Completing the New Host Environment Wizard Page

Completing the New Host Environment Wizard Page

Use the **Completing the New Host Environment Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
New Host Environment Wizard

New Security Policy Wizard

The **New Security Policy Wizard** helps you define how Windows security credentials are established prior to the execution of the server object. The wizard collects information about the following:

- Security policy name
- Source of credentials
- Requirements for host-supplied credentials
- Single Sign-On affiliate applications

In This Section

Welcome to the New Security Policy Wizard Page

Use the **Welcome to the New Security Policy Wizard** page to view the definition of a security policy and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Configure a New Security Policy Wizard Page (in the New Security Policy Wizard) (1)

Configure a New Security Policy Wizard Page (in the New Security Policy Wizard) (1)

Use the first **Configure a New Security Policy** wizard page to identify the security policy and the source of the credentials. Type the name of the security policy, and then select the source for the credentials.

Use this	To do this
Security policy	Type the name for the security policy. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as that of an existing security policy. The name for the new security policy is a required field; you will receive an error message if you click **Next** without providing a name. The name is used to associate the security policy with an object view in other wizard pages and dialog boxes.
Credentials source used to invoke the server	Specify the type or source of credentials that host-initiated processing will associate with the thread when the method on the server object is invoked.
Host-initiated Single Sign-on	Select this option to use host user ID and password. This option is automatically selected if Single Sign-On (SSO) is not installed or not available.
Windows credentials of the HIP application	Select this option to use Windows credentials that are specified on the HIP NT application service. This option is automatically disabled if Single Sign-On (SSO) is not installed or not available.

See Also

Reference

Configure a New Security Policy Wizard Page (in the New Security Policy Wizard) (2)

Configure a New Security Policy Wizard Page (in the New Security Policy Wizard) (2)

Use the second **Configure a New Security Policy** wizard page to identify the source of the host user ID and password and determine how they are translated into Windows-based credentials. This wizard page appears only if **Host-initiated Single Sign-on** is selected on the **Configure a new security policy** wizard page.

Select the source of the credentials and the default user, and then select the credential mapping.

Use this	To do this
Security policy	View the name of the security policy.
Affiliate application	Select the Single Sign-On affiliate application that manages the translation of host user ID and password credentials to Windows credentials.
Single Sign-on mapping use	Specify the credentials to be used when mapping to Single Sign-On.
Default credentials	Select to enter the user ID and password used if the request from the host does not contain a user ID and password or the user ID and password are set to spaces or nulls. When selected, enables the **SSO affiliate application** and **User** boxes. Note This option is disabled if **Use HIP application credentials** is selected on the General tab.
Group application	Select the
User	Type the user name or select a previously entered host user name to be used in the lookup call to SSO.

See Also

Reference

Completing the Security Policy Wizard Page

Completing the Security Policy Wizard Page

Use the **Completing the Security Policy Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
New Security Policy Wizard

Object Wizard (for HIP)

The HIP **Object Wizard** helps you define (or browse for) a Transaction Integrator metadata (.tim) file that is executed by Transaction Integrator after a request is received from a host. The .tim file includes information about the following:

- Windows server object interface, methods, and parameters
- Characteristics of the host application program
- Mappings between the host application program and the Windows server object

If the Windows server object is a .NET Framework object, the wizard also helps you define the assembly and class that implements the interface in the .tim file.

In This Section

Welcome to the Object Wizard Page

Use the **Welcome to the Object Wizard** page to view the definition of an object and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Specify or Locate an Object Wizard Page

Specify or Locate an Object Wizard Page

Use the **Specify or Locate an Object** wizard page to identify objects associated with the application. Select an object from the list, or to create a new object, type the new object name.

Use this	To do this
Object	Select the name of the object. To create a new object, type the new object name. If you select the TIM first, the name is populated with the ProgID or Namespace.Interface defined in the TIM.
Registered TIMs	Type the full path name, including the *.tim extension, to a TIM file, select a recently used file from the list, or click **Browse** and navigate to the file.

See Also

Reference

Define Implementation Characteristics for the .NET Object Wizard Page

Define Implementation Characteristics for the .NET Object Wizard Page

Use the **Define Implementation Characteristics for the .NET Object** wizard page to identify a .NET assembly implementing the interface defined in the Transaction Integrator metadata (TIM) file.

Use this	To do this
Defer implementation identification	Select this option to identify the implementing assembly later through the object's **Properties** dialog.
Pick assembly	Select this option to identify the implementing assembly now.
Path	Type the full path name, including the .dll extension, to a .NET file, select a recently used file from the list, or click **Browse** and navigate to the file.
Class	Select the class containing the methods to be called by HIP runtime for the request processing. The list displays classes from the selected assembly that implement the interface defined in the .TIM file. The first class is selected in the list by default.

See Also
Reference
Completing the Object Wizard Page

Completing the Object Wizard Page

Use the **Completing the Object Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
Object Wizard (for HIP)

New Object View Wizard

The **New Object View Wizard** helps you define the network transport endpoints for the Windows operating system environment. The wizard collects information about the following:

- Local environment name
- Network transport type
- Network transport class
- Endpoint manager
- Endpoint identification

In This Section

Welcome to the New Object View Wizard Page

Use the **Welcome to the New Object View Wizard** page to view the definition of an object view and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Configure a New Object View Wizard Page

Configure a New Object View Wizard Page

Use the **Configure a New Object View** wizard page to select the methods that are included in the object view. Type the name of the new object view, and then select the local environment that accepts requests on behalf of the view.

Use this	To do this
Object view	Type the name of the object view. The name can be a maximum of 259 alpha-numeric characters. The name cannot be the same as that of an existing view.
Local environment	Select a local environment. The names appearing in the list are those local environments available in the **Local Environments** node. The default is the first local environment that appears in the **Local Environments** node.
Host environment	Select a host environment. The names appearing in the list are those host environments available in the **Host Environments** node. Only those host environments that have a **Network Type** that matches the network type defined in the local environments selected in the **Local environment** box are presented for selection. The default is the first local environment that appears in the **Host Environments** node.
Security policy	Select a security policy to be used for determining user credentials.

See Also

Reference

Configure the New View Wizard Page

Configure the New View Wizard Page

Use the **Configure the New View** wizard page to select the methods that are included in the object view. Type the name of the view. To select the methods to be included in the view, select the check box by the method name. Double-click the method name to edit it.

Use this	To do this
View	View the name of the object view.The name of the object view was entered on a previous page of the wizard. To change the name, click **Back**.

Object methods	View all the methods that are defined in the object. Each row in the list contains a summary of the resolution information for a single method. Rows cannot be deleted. The columns are:
	• **Include:** Select this option to include the method in the object view.
	• **Method:** Displays the name of the method. Double-click the method name to display the **Method Resolution Criteria** dialog box.
	• **Endpoint manager:** For a TCP/IP network, select or type the IP address or host name to manage the endpoint. For an SNA network, type the name of the LU alias. In either case, the name can be a maximum of 259 alpha-numeric characters.
	• **Endpoint:** Displays one of the endpoints defined in the local environment associated with the view. To change the content of this field, double-click the **Endpoint** field for a specific method.
	• **Resolution Type:** Displays one of the possible resolution types:
	  • **Endpoint**
	  If the local environment has all of its endpoints in use by object views, associated with the same host environment, that specify **Endpoint Resolution**, you will receive an error message.
	  If the local environment has all of its endpoints in use by object views, associated with the same host environment, that specify **TRM** or **Data** resolution, no endpoint is available for **Endpoint** resolution and **TRM** or **Data** resolution must be supplied for the other methods.
	  If the local environment has one or more endpoints available for endpoint resolution, the first method of the view appears. The first method has its endpoint set to the first endpoint in the local environment that is available for endpoint resolution and the resolution type is set to **Endpoint**. All the remaining methods have the **Endpoint**, **Resolution Type**, **Resolution Data**, and **Resolution Position** boxes cleared.
	  • **Transaction Request Message (TRM)**
	  If the local environment currently has all of its endpoints in use by object views, associated with the same host environment, that specify **Endpoint** or **Data** resolution, the list control is not available.
	  If an endpoint is available for TRM resolution, **Include** is selected for all methods and their endpoint is set to the first endpoint associated with the endpoint that is available for TRM resolution.
	  All methods have resolution type set to **Transaction Request Message (TRM)**; the **Resolution Data** and **Resolution Position** boxes are cleared.
	  • **Enhanced Listener Message (ELM)**
	  • **Data**
	  If the local environment currently has all of its endpoints in use by object views, associated with the same h

ost environment that specified **Endpoint** or **TRM** resolution, the list control is not available.

If an endpoint is available for data resolution, **Include** is selected for all methods and their endpoint is set t o the first endpoint associated with the endpoint that is available for data resolution.

All methods have resolution type set to **Data,** and the **Resolution Data** and **Resolution Position** boxes ar e cleared.

- **Link to Program**

To change the content of this field, double-click the **Resolution Type** field for a specific method.

- **Resolution Data** Identifies the data used to select the method to be executed. To change the content of this field, d ouble-click the **Resolution Data** field for a specific method.

- **Resolution Position** View the resolution position. If **Data Resolution** is selected, this information is used by the TI runtime to identify the starting point in the data stream in which to look for the data defined in the resolution data f ield to determine what method is to be executed. To change the content of this field, double-click the **Resolution P osition** field for a specific method.

See Also

Reference

Method Resolution Criteria Dialog Box (in the New Object View Wizard)

Completing the New Object View Wizard Page

Use the **Completing the New Object View Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also

Reference

New Object View Wizard

Local Environment Dialog Box

Use the **Local Environment** dialog box to select a local environment that is used as a Listener for the application.

Use this	To do this
Local Environment	Select the local environment.

See Also

Other Resources

TI Manager Wizards and Dialog Boxes

Reload TIMs Wizard

The **Reload TIMs Wizard** helps you add multiple Transaction Integrator metadata (TIM) files to the object view at one time. The wizard collects information about the following:

- Name and location of the .tim file
- Availability of the .tim for reloading

In This Section

Welcome to the Reload TIMs Wizard Page

Use the **Welcome to the Reload TIMs** Wizard page to add one or more Transaction Integrator metadata (TIM) files to the object view and to control whether the welcome page is displayed the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also

Reference

Specify or Locate Metadata Files to be Reloaded Wizard Page

Specify or Locate Metadata Files to be Reloaded Wizard Page

Use the **Specify or Locate Metadata Files to be Reloaded** Wizard page to select the Transaction Integrator metadata (TIM) files to be added.

Use this	To do this
Path	Type the full path name, including the .tim extension, to a TI metadata file, select a recently used file from the list, or click **Browse** and navigate to the file. Only .tim files that are not currently in use by other object views are displayed. The path can be a maximum of 255 Unicode characters, and the name of the .tim file must be unique.

See Also

Reference

Reloading of Metadata Files Wizard Page

Reloading of Metadata Files Wizard Page

Use the **Reloading of Metadata Files** Wizard page to view the Transaction Integrator metadata (TIM) files that will be reloaded. The Wizard analyzes the .tim files selected on the previous page to verify the number of methods, as well as the name and ID of each method.

Use this	To do this
Messages	View the status of each .tim file. If a file is acceptable for reloading, the file name is followed by **OK**. If a file is not acceptable for reloading, the file name if followed by a message indicating the reason for being rejected. Note The files given an **OK** are not actually reloaded until you click **Next**.

See Also

Reference

Completing the Reload TIMs Wizard Page

Completing the Reload TIMs Wizard Page

Use the **Completing the Reload TIMs Wizard** page to review the .tims you selected in the previous wizard pages. You can return to an earlier wizard page to select a different .tim by clicking **Back**.

See Also
Reference
Reload TIMs Wizard

New Remote Environment Wizard

In This Section

Welcome to the New Remote Environment Wizard Page

Use the **Welcome to the New Remote Environment Wizard** page to view the definition of a remote environment and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Configure a New Remote Environment Wizard Page

Configure a New Remote Environment Wizard Page

Use the **Configure a New Remote Environment** wizard page to identify the name and location of the remote environment. To define the remote environment, type the name of the remote environment, and then select the network type.

Use this	To do this
Name	Type the name for the remote environment. The name can be a maximum of 256 Unicode characters. The name cannot be the same as that of an existing remote environment. The name for the new remote environment is a required field; you will receive an error message if you click **Next** without providing a name.
Vendor	Select the vendor supplying the remote environment.
Network type	Select the type of network used to communicate with the remote environment. The options are: • **TCP/IP** (default) • **SNA**

See Also

Reference

Configure Host Environment and Programming Model Wizard Page

Configure Host Environment and Programming Model Wizard Page

Use the **Configure Host Environment and Programming Model** wizard page to identify information about the target host configuration and show where the host's programming model is defined.

Use this	To do this
Target host	Select a target host. For the TCP Network type, the available choices are: • **CICS** • **IMS** • **OS400** For the LU 6.2 (SNA) for the network type, the available choices are: • **CICS** • **IMS**

Programming model	Select a programming model. For the TCP network type, the available choices are:
	• For the CICS host
	  • **TCP TRM User Data**
	  • **TCP ELM User Data**
	  • **TCP TRM Link**
	  • **TCP ELM Link**
	• For the IMS host
	  • **IMS Connect**
	  • **IMS Implicit**
	  • **IMS Explicit**
	• For the OS400 host
	  • **Distributed Program Call**
	
	For the LU 6.2 (SNA) network type, the available choices are:
	• For the CICS host
	  • **CICS LU 6.2 User Data**
	  • **CICS LU 6.2 Link**
	• For the IMS host
	  • **IMS LU 6.2 User Data**

See Also

Reference

Configure Endpoint TCP/IP Wizard Page

Configure Endpoint TCP/IP Wizard Page

Use the **Configure Endpoint TCP/IP** wizard page to set the IP address and port number for the host environment.

Use this	To do this
IP/DNS Address	Type the address or select from the drop-down list. The IP address can be defined either by name or numerical address. The address can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used IP addresses.
Port list	Select the port number from the drop down list. Click **Edit** to edit the selected port list, or create a new one by selecting <**new port list**> and clicking **Edit**.

See Also

Reference

Completing the New Remote Environment Wizard Page

Port List Editing Dialog Box

The **Port List Editing** dialog box allows you to create new TCP ports or edit existing ports.

To add a port to addition port list, select the port list on the previous panel, press edit.

The Defined ports will serially list the defined ports within the list. Highlight a port within the defined list, press remove to remove the port from the list.

To add a new port to the list, type the port number within the new port text box, press add. The port number will be added to the defined port list. Press OK, the new port will show up within the port list separated by semicolons within the port list.

To create a new port list, from the port list drop down menu, select <new port list>, press edit.

Use this	To do this
New port	Type the number of the new port. The port number can be a maximum of 256 positive numerals separated by semicolons. A port number can be defined with up to 5 positive digits, and more than one port can be defined to an address. The port number cannot be larger than 32767. The port numbers cannot be repeated.
Add	Click to add the new port number to the list of defined ports.
Remove	Click to remove the selected port number from the list.
Defined ports	View the port numbers already defined.

See Also

Reference

Completing the New Remote Environment Wizard Page

Configure Endpoint SNA Wizard Page

Use the **Configure SNA Endpoint** wizard page to set the local LU alias, the remote LU alias, and the APPC mode name for the host environment.

Use this	To do this
Local LU alias	Type the local LU alias. The alias can be a maximum of 256 alphanumeric characters. The drop-down list displays all the Local APPC LUs defined to the SNA Manager.
Remote LU alias	Type the remote LU alias. The alias can be a maximum of 256 alphanumeric characters. The drop-down list displays all the remote APPC LUs defined to the SNA Manager.
Mode name	Type the APPC mode used for the connection. You can obtain a list of configured modes from Host Integration Server **SNA Manager**. Note To use two-phase commit, the APPC mode must be Sync Level 2 capable.

See Also

Reference

Completing the New Remote Environment Wizard Page

Configure Endpoint IMS Connect Wizard Page

Use the **Configure Endpoint IMS Connect** wizard page to set the IP address and port number for the host environment and to identify the IBM-defined exit program on the host.

Use this	To do this
IP/DNS Address	Type the address or select from the drop-down list. The address can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used IP addresses.
Port list	Type the TCP port number or select from the drop-down list. The number can be a maximum of 256 positive numerals separated by semicolons. The port numbers cannot be repeated. The dropdown list displays the most recently used port numbers.
ITOC information	Select either **HWSIMSO0** or **HWSIMSO1**, the IBM-defined exit associated with IMS connect on the host.
Format to use	View the choice of formats.
HWSIMSO0	Select this option to use the HWSIMSO0 exit program.
HWSIMSO1	Select this option to use the HWSIMSO1 exit program.
IMS system ID	Type the IMS system ID or select from the drop-down list. The ID can be a maximum of 8 alphanumeric characters. The drop-down list displays the most recently used IMS system IDs.
ITOC exit name	Type the ITOC exit name or select from the drop-down list. The name can be a maximum of 8 alphanumeric characters. The drop-down list displays the most recently used ITOC exit names.

See Also

Reference

Completing the New Remote Environment Wizard Page

Configure Endpoint Diagnostic Capture Wizard Page

Use the **Configure Diagnostic Capture Endpoint** wizard page to

Use this	To do this
Recording file	Type the name of the recording file or select from the drop-down list. The file name can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used recording files.
Use responses from the remote environment	Select this option to specify a remote environment. When the option is selected, the drop-down is enabled and displays all the remote environments that have the same network type, host environment, and programming model as the remote environment being defined.
Name	Type the name of the remote environment or select from the drop-down list. The name can be a maximum of 256 alphanumeric characters.

See Also

Reference

Completing the New Remote Environment Wizard Page

Configure Endpoint Playback Wizard Page

Use the **Configure Playback Endpoint** wizard page to

Use this	To do this
Recording file	Type the name of the recording file or select from the drop-down list. The file name can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used recording files.

See Also

Reference

Completing the New Remote Environment Wizard Page

Completing the New Remote Environment Wizard Page

Use the **Completing the New Remote Environment Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
New Remote Environment Wizard

Object Wizard (for WIP)

In This Section

Welcome to the Object Wizard Page

Use the **Welcome to the Object Wizard** page to view the definition of a metadata file (.tlb for COM, .dll for .Net) and to control whether the welcome page is displayed again the next time the wizard is used.

Use this	To do this
Do not show this welcome page again.	Select this option if you will be using the wizard more than once and do not want to read the introduction each time the wizard is launched. Each time the wizard is launched, the welcome page will be hidden and the wizard will begin with the second page. To view the welcome page again, click **Back** on the second page of the wizard.

See Also
Reference
Specify or Locate an Object Wizard Page

Specify or Locate an Object Wizard Page

Use the **Specify or Locate an Object** wizard page to specify or initiate a browse operation to locate the Transaction Integrator metadata file that represents the server object. For COM based interfaces, the file extension is .tlb, for .Net the file extension is .dll.

Use this	To do this
Path	Type the full path name to a metadata file. If the file does not end with the .tlb or .dll extension, an error message will appear. You can add more than one object at a time.
Browse	Launches the **Open** dialog box.
Remove	Removes the selected objects from the list.

See Also

Reference

Define Environment Characteristics for the COM Object Wizard Page

Define Environment Characteristics for the .NET Object Wizard Page

Define Environment Characteristics for the COM Object Wizard Page

Use the **Define Environment Characteristics for the COM Object** wizard page to define the COM+ application in which Transaction Integrator should run.

Use this	To do this
COM+ applications	Select a COM application. The contents of this box are populated from enumerating the COM+ applications on the local computer. To appear in this list, the application should have Server Activation type and be editable (that is, **Disable Changes** should not be selected on the application's **Advanced Properties**).

See Also

Reference

Define Remote Environment Wizard Page

Define Environment Characteristics for the .NET Object Wizard Page

Use the **Define Environment Characteristics for the .NET Object** wizard page to define the IIS Virtual Directory in which Transaction Integrator should run.

Use this	To do this
Self-hosted	Select this to reference the TI assembly within the application itself. No IIS security will be available, and the ot her options on this page will be unavailable.
ASP .NET wor ker process	Select this to use IIS security.
Virtual direct ories	Select an IIS virtual directory. The list is populated from enumerating the IIS Virtual Directories on the local co mputer. Note that if you do not choose an environment, an error appears when you click **Next**. If there is no environm ent available, you must exit this wizard and define the appropriate environment.
Service	Choose **Remoting only**, **Remoting and Web Service**, or **Web Service only**.
Key file for pr oxy	Select the key (.snk) file that TI Manager uses to sign the proxy in the IIS virtual directory. The list is populated with files used previously.

See Also

Reference

Define Remote Environment Wizard Page

Define Remote Environment Wizard Page

Use the **Define Remote Environment** wizard page to select a remote environment (RE) that provides the connection information to the host application. You can select the RE from the list of already defined remote environments that are of the appropriate type (that is, the network, host environment, and programming model of the remote environment must agree with those of the object).

Use this	To do this
Remote environments	Select a remote environment. The contents of this list are populated from an enumeration of the remote environments already defined which match in network type, host environment and programming model.

See Also

Reference

Completing the Object Wizard Page

Completing the Object Wizard Page

Use the **Completing the Object Wizard** page to review the choices and settings you made in the previous wizard pages. You can return to an earlier wizard page to change a setting by clicking **Back**.

See Also
Reference
Object Wizard (for WIP)

Import WIP Definitions Wizard

Use this wizard to import WIP definitions.

In This Section

Welcome to the Import WIP Definitions Wizard Page

Click **Next** to continue.

Define Import Characteristics Page

Use this page to select a directory containing the exported definitions.

Duplicate Remote Environment Use

Select Original or Exported Definitions.

Import Directory Path

Type or browse to the desired directory.

Importing WIP Definitions Page

This page displays errors and other information during the import process.

Click **Next** when the process is complete.

Import WIP Definitions Wizard Finish Page

This page appears when the import process is complete.

Click **Finish** to close the wizard.

Export WIP Definitions Wizard

Use this wizard to export WIP definitions.

In This Section

Welcome to the Export WIP Definitions Wizard Page

Click **Next** to continue.

Define Export Characteristics Page

Directory use

Select **Use HIS Generated Directory** to have Transaction Integrator create a target directory.

Select **Use Specific Directory** to browse to a directory of your choice.

Remote Environment Selection Page

Specify the Remote Environments to be exported, and for each one whether Connection Information should be retained.

WIP Object Selection Page

Specify the WIP Objects to be exported.

Exporting WIP Definitions Page

You can view any relevant information in the **Messages** box.

Export WIP Definitions Wizard Finish Page

Click **Finish** to close the wizard.

TI Manager Properties

Transaction Integrator (TI) Manager provides property pages for viewing and setting properties. The property pages include the following:

Transaction Integrator (Configuration) Properties

- Timeout Tab

Application Properties

- General Tab (Application Properties)
- Advanced Tab (Application Properties)
- .NET Assembly Path Tab (Application Properties)

Listener Properties

- General Tab (Listener Properties)
- Endpoints Tab (TCP/IP Listener Properties)
- Endpoints Tab (SNA Listener Properties)
- Application Tab (Listener Properties)

Local Environment Properties

- General Tab (Local Environment Properties)
- Endpoints Tab (TCP/IP Local Environment Properties)
- Endpoints Tab (SNA Local Environment Properties)

Host Environment Properties

- General Tab (Host Environment Properties)
- Network Tab (Host Environment Properties)
- Conversion Tab (Host Environment Properties)
- Default Tab (Host Environment Properties)

Security Policy Properties

- General Tab (Security Policy Properties)
- Credentials Source Tab (Security Policy Properties)

Object Properties

- General Tab (Object Properties)

- Methods Tab (Object Properties)
- .NET Implementation Tab (Object Properties)

Object View Properties

- General Tab (View Properties)
- Host Environments Tab (View Properties)
- Methods Tab (View Properties)
- Method Resolution Criteria Dialog Box (Object Properties)

Remote Environment Properties

- General Tab (Remote Environment Properties)
- TCP/IP Tab (Remote Environment Properties)
- LU6.2 Tab (Remote Environment Properties)
- Target Tab (Remote Environment Properties)
- Recording Tab (Remote Environment Properties)
- Locale Tab (Remote Environment Properties)
- Security Tab (Remote Environment Properties)
- CICS Tab (for LU6.2 Link Properties)
- CICS Tab (for TCP/IP Properties)
- IMS Tab
- SSO tab (Security Policy Properties)

Object Properties

- General Tab (Object Properties)
- Hosting Tab (COM Object Properties)
- Remote Environment Tab (Object Properties)
- Security Tab (Remote Environment Properties)

Timeout Tab

Use the **Timeout** tab to set the maximum number of minutes the TI Manager console is allowed to be inactive while it is in **Configuration** mode.

Use this	To do this
Write mode timeout	Type the maximum number of minutes the TI Manager console can remain in configuration mode without a keystroke or mouse movement that changes the configuration. At the end of this time, the console returns to **View and operate** mode. The default is **30** minutes.

⚠**Caution**
The properties of the TI Manager console are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the console to function incorrectly.

See Also

Reference

Transaction Integrator (mode) Node

General Tab (Application Properties)

The **General** tab allows you to set the general properties of the application.

Use this	To do this
Application	View the name of the application.
Computer	View the name of the computer running the application.
Process ID	View the process ID of the HIPService. **0** indicates that HIPService is not running.
Worker threads	Type the number of processing threads to be used in the work of the application. The number of threads must be greater than 0 and fewer than 257.
Queued requests	Type themaximum number of requests that can be stored in the queue before new requests are rejected. The number of queued requests must be equal to or greater than the number of worker threads, and must be less than 2049.
Comment	Type additional information about the use of the object view in the HIP environment. The comment can be a maximum of 259 alpha-numeric characters.
Service starts automatically	Select this option to start the HIPService automatically at startup. Clear the check box to require manual start up.

⚠**Caution**
The properties of an application are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Application

Advanced Tab (Application Properties)

The **Advanced** tab allows you to set the advanced properties of the application.

Use this	To do this
Minimum active worker threads	Type or select the minimum number of worker threads that the HIP runtime maintains. The number of threads must be greater than 0 and fewer than 193. The number cannot exceed the number of **Worker threads** set on the **General** tab property page. The default is **8** threads.
Cleanup worker threads	The group of properties controlling thread cleanup.
When no requests have been queued for	Type or select the number of seconds the HIP runtime will wait after the last client request has been received before it terminates the threads. The minimum wait is 1 second and the maximum wait is 3600 seconds. The default is **30** seconds.
Start an additional working thread	The group of properties controlling when new threads are started.
When more than *number* requests have been queued	Type or select the number of requests that must be received before additional worker threads are started. The minimum number of requests is 1, and the maximum number of requests is 99,999. The default is **15** requests. The number of requests cannot exceed the **Maximum queued requests** (set on the **General** tab) multiplied by the **During the previous *number* seconds** (set on this tab).
During the previous *number* seconds	Type or select the number of seconds the HIP runtime must wait before starting additional worker threads. The minimum number of seconds is 1, and the maximum number of seconds is 3600. The default is **5** seconds.
Increase context cache size	The group of properties controlling the size of the cache of incoming context requests over a period of time.
When more than *number* contexts have been created	Type or select the number of contexts that must be created before the HIP runtime increases the size of the cache. The request context is needed to pass the parameters of the request from the listener to a worker thread. The initial cache size is equal to the number of worker threads multiplied by 2, which is normally less than the maximum queue length. If the cache is empty, a new context is created but will be destroyed at the end because there is no place in the cache to put it in. Increasing the cache size allows the HIP runtime to save the context for future reuse to eliminate expensive operations of creating and releasing the context. The cache size can grow up to the maximum queue length. The minimum number of contexts is 1, and the maximum number of contexts is 192. The default is **30** contexts.

During the previous *number* seconds	Type or select the number of seconds the HIP runtime must wait before increasing the cache size. The minimum number of seconds is 1, and the maximum number of seconds is 3600. The default is **5** seconds.

⚠Caution

The properties of an application are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also
Reference
Application

.NET Assembly Path Tab (Application Properties)

The **.NET Assembly Path** tab allows you to set the way the application locates the implementation assemblies.

Use this	To do this
Use GAC	Select this option to search for the assemblies in the Global Assembly Cache.
Path	Type the full path name of the folder that contains implementation assemblies, or select a recently used folder from the list.

⚠Caution

The properties of an application are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Application

General Tab (Listener Properties)

The **General** tab displays the general properties of the application's local environment. The controls on the **General** tab are read-only.

Use this	To do this
Name	View the name of the local environment. To change the name of the local environment, right-click the **Local Environments** node.
Network type	View the type of network used to communicate with the host.
Transport class	View the class of the transport object.
Comment	View a comment about the local environment. To change an existing comment, right-click the **Local Environments** node.

Note

When viewed from the ***listener*** node, the general properties are read-only. If you want to change a general property, right-click the specific ***local environment*** **Node** under the **Local Environments Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties of an listenerare not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Listener

Endpoints Tab (TCP/IP Listener Properties)

The **Endpoints** tab displays the defined endpoints in the local environment definition. The controls on the **Endpoints** tab are read-only. To add or remove endpoints from the local environment, use the **Properties** shortcut menu item on the **Local Environments** node.

Use this	To do this
New port or service name	Not used.
Defined ports or service names	View the valid ports and services names for the local environment.

Note

When viewed from the ***listener*** node, the endpoint properties are read-only. If you want to change an endpoint property, right-click the specific ***local environment*** **Node** under the **Local Environments Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties of an application are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Listener

Endpoints Tab (SNA Listener Properties)

The **Endpoints** tab displays the defined endpoints in the local environment definition. The controls on the **Endpoints** tab are read-only. To add or remove endpoints from the local environment, use the **Properties** shortcut menu item on the **Local Environments** node.

Use this	To do this
New transaction program name	View the transaction program (TP) name or the link mirror transaction ID of the local environment.
Add	Not available.
Defined transaction programs names	View the existing TP names or the link mirror transaction IDs for the local environment.
Remove	Not available.

Note

When viewed from the ***listener*** node, the endpoint properties are read-only. If you want to change an endpoint property, right-click the specific ***local environment*** **Node** under the **Local Environments Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties of an listenerare not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Listener

Application Tab (Listener Properties)

The **Application** tab displays the local environment properties that are specific to the application. It allows you to specify whether the application will begin listening on the local environment endpoints automatically when the application is started.

Use this	To do this
Begin Listening on the local environment at application startup	When this option is selected, the application will automatically start listening for incoming host requests on each port defined in the local environment.

⚠**Caution**
The properties of an listener are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

Listener

General Tab (Local Environment Properties)

Use the **General** tab to set the general local environment properties, including its name and basic information about its network type.

Use this	To do this
Local environment	Type the name for the local environment. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as that of an existing local environment. You can change the name at any time, unless the local environment is an associated listener in an application that is active. The application must be stopped before the LE can be renamed. After the name is changed, the new name will appear in the **Application**, **Local Environments** and **Object** nodes.
Network type	Select the type of network used to communicate with the host. The choices are: • TCP/IP (default) • SNA. Note If the local environment is associated with any other configuration entity, you will receive an error when you attempt to apply the change.
Transport class	Select the class of the transport object. This list changes according to the selected **Network type**.
Endpoint manager	Select or type the local LU alias to manage the endpoint. Note The **Endpoint manager** edit box is set to **Localhost** and disabled if the **Network type** is set to **TCP/IP**.
Comment	Type any additional information about the local environment. The comment can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special).

Caution
The properties of a local environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the local environment to function incorrectly.

See Also

Reference

Local Environments Node

Local Environment Node

Endpoints Tab (TCP/IP Local Environment Properties)

Use the **Endpoints** tab to view, add, and delete endpoints in the local environment definition.

Use this	To do this
New port or service name	Type the port number or service name for the local environment. The port number can be a maximum of 5 numeric characters and must be greater than 0 and fewer than 32768. The service name can be a maximum of 64 alpha-numeric characters. The format of the name must conform to the WinSock Service Name convention. The port number or name cannot be the same as that of an existing number or name in **Defined ports and service names**.
Add	Click to validate the port number or service name you typed. If the number or name is valid, it then appears in **Defined ports and service names**. If the local environment is referenced by an application, the new port or service name will not be available for use until all the applications that use the local environment are restarted.
Defined ports and service names	View the valid ports and services names for the local environment.
Remove	Click to remove a selected port number or service name from **Defined ports or service names**.

⚠**Caution**
The properties of a local environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the local environment to function incorrectly.

See Also

Reference

Local Environments Node

Local Environment Node

Endpoints Tab (SNA Local Environment Properties)

Use the **Endpoints** tab to view, add, and delete endpoints in the local environment definition.

Use this	To do this
New transaction program name	Type the transaction program (TP) name or the link mirror transaction ID of the local environment. The name or ID can be a maximum of 64 alphabetic characters. The format of the name must conform to the IBM SNA Transaction Program Names convention. The name or ID cannot be the same as that of an existing name or ID in the **Defined transaction programs names**.
Add	Click to validate the transaction program (TP) name or the link mirror transaction ID you typed. If the TP name or the link mirror transaction ID is valid, it appears in **Defined transaction program names**.
Defined transaction programs names	View the existing TP names or the link mirror transaction IDs for the local environment.
Remove	Click to remove one or more TP names or link mirror transaction IDs selected from the **Defined transaction programs names** available for use in the local environment.

Caution
The properties of a local environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the local environment to function incorrectly.

See Also

Reference

Local Environments Node

Local Environment Node

General Tab (Host Environment Properties)

The **General** tab defines the basic characteristics of the host environment that will initiate requests to the Windows platform.

Use this	To do this
Host environment	Type the name for your host environment. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as that of an existing host environment.
Time-out in seconds	The **Send** and **Receive** time-out values are used by the host-initiated processing (HIP) runtime environment when it communicates with the host environment. The time-out values are used on transport-specific application program interfaces (APIs) to terminate the send and/or receive API functions if the expected data is not received in the specified amount of time.
Send	Type the number of seconds the HIP runtime should wait for an acknowledgement before it times out. The maximum number of seconds is 3,600; the default is **30** seconds.
Receive	Type the number of seconds the host should wait for a response before it times out. The maximum number of seconds is 3,600; the default is **30** seconds.
Comment	Type any additional information about the host environment. The comment can be a maximum of 259 alpha-numeric characters.

⚠**Caution**
The properties of a host environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the host environment to function incorrectly.

See Also

Reference

Host Environments Node

Host Environment Node

Network Tab (Host Environment Properties)

The **Network** tab defines the network characteristics that the host uses to interact with the Windows environment and to ensure that unauthorized requests are not inadvertently processed.

Use this	To do this
Network type	Select the type of network used to communicate with the host environment. The choices are: • TCP/IP (default) • SNA
Remote endpoint manager	For a TCP/IP network, select or type the IP address of the host or the host name registered in the DNS. For an SNA network, type the LU name associated with the host system. In either case, the name can be a maximum of 259 alpha-numeric characters.

Caution
The properties of a host environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the host environment to function incorrectly.

See Also

Reference

Host Environments Node

Host Environment Node

Conversion Tab (Host Environment Properties)

The **Conversion** tab defines the data type characteristics used by the host that makes requests to the Windows platform. The code page and elemental data conversion object are used by host-initiated processing (HIP) to transform the incoming and outgoing data to a form that can be used by the host application program.

Use this	To do this
Data conversion	Select the data conversion routine to be used.
Host code page	Select the code page used by the host. Each available code page is listed by its symbolic value.

Caution
The properties of a host environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the host environment to function incorrectly.

See Also

Reference

Host Environments Node

Host Environment Node

Default Tab (Host Environment Properties)

The **Default** tab of the properties page of a specific host environment defines the manner in which the host typically communicates with Transaction Integrator. These defaults are applied during the configuration for an object view.

Use this	To do this
Default method resolution criteria	View the definition of the default behavior for setting up the method resolution criteria for a new object view.
Type	Select the type of resolution to be conducted: **Endpoint**: The endpoint resolution type is the simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables the **Object**, **Input format** and **Output format** boxes and clears their contents. • **Transaction Request Message (TRM)**: The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant, MSLink model. It represents a double exchange sequence. The first exchange represents the transaction request; the second exchange represents the request and reply data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Transaction Request Message (TRM)** enables the **Object**, **Input format**, and **Output format** boxes. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting one of the predefined **Transaction Request Objects** disablesthe **Input format** and **Output format** boxes; it also disables the **Link Tran ID** box and clears its contents. Selecting a custom object enables the **Input format** and **Output format** boxes and sets their initial values to the Microsoft-supplied TRM handler items. • **Data**: Similar to **Endpoint** in that it adds the ability to identify a string in the data stream directed to a port, that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data. Selecting **Data** disables the **Object**, **Input format**, **Output format,** and **Link Tran ID** boxes and clears their contents. • **Link:** The Link resolution type is specific to the IBM CICS DPL model. The endpoint name is representative of the TP name that is the CICS mirror transaction ID. Select **Link** to resolve all the methods of a new object view using the data information specified when the object view was created. When a new view is created and associated with the HE, the method resolves to the Link Tran ID field in the default HE resolution. Selecting **Link** disables the **Object**, **Input format**, and **Output format** boxes and clears their contents.

Object	Select the object. The list of objects is populated from the Help string in the interface definition of objects identified to the resolution handler. Microsoft provides three TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS** • **Microsoft HIS - Link Handler**
Input format	Select the format for the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is populated from the user-defined type definitions in the MicrosoftTRMDef.tim file. The **Input format** box is disabled for standard TRM handlers and enabled for custom TRM handlers, but three input formats are defined that match the names of the resolution handler **Object**: • **TRMINMSLink** • **TRMINMSCCS** • **TRMINIBMCCS**
Output format	Select the format for the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream to be returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is populated from the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is disabled for standard TRM handlers and enabled for custom TRM handlers, but three output formats are defined that match the names of the resolution handler **Object**: • **TRMOUTMSLink** • **IBM Listener Reply Format (TRMOUTCCS)** • **Link Tran ID:** Select the TP name or Link Mirror Tran ID for the SNA local environment. This control is enabled only when the **Network Type** is set to SNA and the **Resolution Type** is set to Link.

⚠Caution

The properties of a host environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the host environment to function incorrectly.

See Also

Reference

Host Environments Node

Host Environment Node

General Tab (Security Policy Properties)

The **General** tab of the properties page of a security policy defines the name of the policy and the source of the security credentials.

Use this	To do this
Policy	Type the name for the security policy. The name can be a maximum of 259 Unicode characters (alphabetic, numeric, space, and special). The name cannot be the same as the name of an existing security policy. The name for the new security policy is a required field; you will receive an error message if you click **Next** without providing a name. The name is used to associate the security policy with an object view in other wizard pages and dialog boxes.
Host-initiated Single Sign-on	Select this option to use host user ID and password. This option is automatically disabled if Single Sign-On (SSO) is not installed or not available.
Windows credentials of the HIP application	Select this option to use Windows user ID and password specified on the HIP NT application service. This option is automatically selected if Single Sign-On (SSO) is not installed or not available.
Comment	Type additional information about the security policy. The comment can be a maximum of 259 Unicode characters.

⚠**Caution**
The properties of a security policy are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the security policy to function incorrectly.

See Also

Reference

Security Policies Node

Security Policy Node

Credentials Source Tab (Security Policy Properties)

The **Credentials Source** tab of the properties page of a security policy defines the source of the host user ID and password and describes how they are translated into Windows-based credentials.

Use this	To do this
Security policy	View the name of the security policy.
Affiliate application	Select the SSO affiliate application to be queried to gain access to the Windows credentials needed to execute methods on the server object. The list displays all the affiliate applications defined in the SSO. The display name of the affiliate application is a concatenation of the affiliate application name and description, separated by a hyphen. The selected application will be added to the security policy. Note The dropdown list is disabled if SSO is not installed or is not available.
Single Sign-on mapping use	Specify the credentials to be used when mapping to Single Sign-on.
Default credentials	Select this option to enter the user ID and password used if the request from the host does not contain a user ID and password or the user ID and password are set to spaces or nulls. This option enables the **Group application** and **User** boxes.
Group application	The default user group provides the default user ID and password.
User	Type or select the host user name used in the lookup call to SSO. Note The dropdown list is disabled if SSO is not installed or is not available.

Caution
The properties of a security policy are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the security policy to function incorrectly.

See Also

Reference

Security Policies Node

Security Policy Node

Method Resolution Criteria Dialog Box (in the New Object View Wizard)

Use the **Method Resolution Criteria** dialog to provide detailed information about a specific method on the object view.

Use this	To do this
Method	View the name of the object's method for which the method resolution criteria are being set.
Endpoint	Select the endpoints on the local environment that are available for the type of resolution specified in **Type**. All endpoints in the LE are listed and available for selection. If an endpoint selected for one resolution type is already used with another resolution type then an error message will pop up when you click **Apply**. The error message explains which object and view has this endpoint used with another resolution type.
Method resolution criteria	View a description of the behavior for the IT runtime when it resolves a request to a method. The sequence of data flow events may require the TI to send and receive various amounts of data prior to obtaining the ability to make the connection between the incoming request and the target method.
Type	Select the type of resolution to be conducted: • **Endpoint** The endpoint resolution type is simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and it represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables **Object Name**, **Input format** and **Output format** and clears their contents. • **Transaction Request Message (TRM)** The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant MSLink model. It represents a double exchange sequence. The first exchange represents the transaction request, and the second exchange represents the request and reply data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Transaction Request Message (TRM)** enables **Object**, **Input format**, and **Output format**. This option is available only for TCP/IP local environment/host environment pairs. • **Enhanced Listener Message (ELM)**: • **Data** Similar to **Endpoint** in that it adds the flexibility to identify a string in the data stream directed to a port that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Data** disables **Object**, **Input format**, and **Output format** and clears their contents. • **Link to Program Name:** Resolves all the methods of a new object view using the data information specified when the object view was created. When a new view is created and associated with the HE, the method resolves to the Link Tran ID field in the default HE resolution. Selecting **Link to Program Name** disables **Object**, **Input format**, and **Output format** and clears their contents. This option is available only for SNA local environment/host environment pairs. Note Only **TRM-MSLink** and **Link to Program Name** are available for Link methods. Methods can be defined as Link in Visual Studio by setting the **Is Link** property to **True**.

Object Name	Select the object. The list of objects is populated from the Help string in the interface definition of objects identified to the resolution handler. Microsoft provides three TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS**
Input format	Select the format of the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is determined by the UDT) definitions in the MicrosoftTRMDef.tim file in the installation TIMLibs directory. The **Input format** box is blank by default, but three input formats are defined that match the names of the resolution handler **Object**: • **MS Link Request Format** • **IBM Listener Type 1 Request Format** • **IBM Listener Type 2 Request Format**
Output format	Select the format of the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream that is returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is determined by the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is blank by default, but three output formats are defined that match the names of the resolution handler **Object**: • **MS Link Reply Format** • **IBM Listener Type 1 Reply Format** • **IBM Listener Type 2 Reply Format**
Link	View the name of the program being linked to.
Program Name	Name of the TP program.
Resolution Data	View a definition of the data used by the IT runtime when it resolves a request to a method.
Data	Type the method resolution criteria for a method on the object view. The criteria can be a maximum of 256 alpha-numeric characters. This control is enabled for resolution types **Transaction Request Message (TRM)** and **Data**. This control is disabled for resolution type **Endpoint**.
Position	Type the resolution position. The position can be a maximum of nine numeric characters. This control is enabled for resolution type **Data** and disabled for resolution types **Transaction Request Message (TRM)** and **Endpoint**.

See Also

Other Resources

TI Manager Properties

General Tab (Object Properties)

Use the **General** tab to view the properties of an object. All properties are read-only, except for the **Object** and **Comment** fields.

Use this	To do this
Object	View the name of the object.
Metadata file	View the name of the Transaction Integrator metadata (.tim) file.
Method count	View the number of methods that are contained in the object definition.
Help string	View the Interface Definition Help String that was part of the object's interface definition.
Comment	Type a comment that provides additional information about the use of the object in host-initiated processing. The comment can be a maximum of 256 alpha-numeric characters.

Caution

The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (HIP)

Object Node (HIP)

Methods Tab (Object Properties)

Use the **Methods** tab to view the methods on an object.

Use this	To do this
Method	View the name of the methods defined in the .tim file.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (HIP)

Object Node (HIP)

.NET Implementation Tab (Object Properties)

Use the **.NET Implementation** tab to view the properties of the .NET assembly implementing the interface defined in the Transaction Integrator metadata (TIM) file.

Use this	To do this
Assembly path	Type the full path name, including the .dll extension, to a .NET file, select a recently used file from the list, or click **Browse** and navigate to the file.
Class	Select the class containing the methods to be called by HIP runtime for the request processing. The list displays classes from the selected assembly that implement the interface defined in the .TIM file. The first class is selected in the list by default.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (HIP)

Object Node (HIP)

General Tab (View Properties)

Use the **General** tab to view or change the basic information about the object view.

Use this	To do this
Object view	View the name of the object view.
Local environment	Select a local environment (LE). The LE determines the network type associated with the object view. If you are changing from one network type to another, be sure to remove all methods from the view before changing the LE. Note Changing the current LE in the view removes the host environments (HEs) associated with the view and all related information. Use the **Host Environments** tab to associate a new HE.
Security policy	Select a security policy to be associated with the view.
Comment	Type a comment that provides additional information about the use of the object view in the HIP environment. The comment can be a maximum of 259 alpha-numeric characters.

Note

When viewed from the ***view*** node under the application ***listener*** node, the general properties are read-only. If you want to change a general property, right-click the specific ***view*** **Node** under the ***object*** **Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties on an object view are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the application to function incorrectly.

See Also

Reference

View Node (listener)

View Node (object)

Host Environments Tab (View Properties)

Use the **Host Environments** tab to view or change detailed information about the host environments (HEs) associated with the object view.

Use this	To do this
Host environment	Select the host environment to associate with the object view. This list includes all the HEs that have the same network type as the local environment associated with the object view.
Configured host environments	View the HEs already associated with the object view.
Add	Click to add the HE to the object view and move the name to **Configured host environments**.
Remove	Click to remove the selected HE from the object view and from **Configured host environments**. The removed HE is moved to the list in **Host environment**.

Note

When viewed from the ***view*** node under the application ***listener*** node, the host environment properties are read-only. If you want to change a host environment property, right-click the specific ***view*** **Node** under the ***object*** **Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties on an object view are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object view to function incorrectly.

See Also

Reference

View Node (listener)

View Node (object)

Methods Tab (View Properties)

Use the **Methods** tab to view or change the methods associated with the object view and the method resolution details. You can use this page to add and remove methods on the object view. Double-click a row in the grid to edit the resolution information.

Use this	To do this

Object methods	View all the methods that are defined in the object. Each row contains a summary of the resolution information for a single method. Rows in the list cannot be deleted. The columns are:
	• **Include** Select to include the method in the object view.
	• **Method** View the name of the method. Double-click the method name to launch the **Method Resolution Criteria** dialog box.
	• **Endpoint** View one of the endpoints defined in the local environment associated with the view. To change the content of this field, double-click on the **Endpoints** field for a specific method.
	• **Resolution Type** Displays one of the possible resolution types:
	  • **Endpoint** The endpoint resolution type is the simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and represents a single synchronous exchange of data.
	  • **Transaction Request Message (TRM)** The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant, MSLink model. It represents a double exchange sequence.
	  • **Enhanced Listener Message (ELM)**:
	  • **Data** Similar to **Endpoint** in that it adds the ability to identify a string in the data stream directed to a port that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data.
	  • **Link to Program Name:** Select **Link to Program Name** to resolve all the methods of a new object view using the data information specified when the object view was created.
	To change the content of this field, double-click the **Resolution Type** field for a specific method.
	• **Resolution Data** Identifies the data used to select the method to be executed. To change the content of this field, double-click the **Resolution Data** field for a specific method.
	• **Resolution Position** Displays the resolution position. If **Data Resolution** is selected, this information is used by the TI runtime to identify the starting point in the data stream in which to look for the data defined in the resolution data field to determine which method is to be executed. To change the content of this field, double-click on the **Resolution Position** field for a specific method.

Note

When viewed from the ***view*** node under the application ***listener*** node, the method properties are read-only. If you want to change a method property, right-click the specific ***view*** **Node** under the **object Node**, and then left-click **Properties** on the shortcut menu. The properties on that page are read-write.

Caution

The properties on an object view are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object view to function incorrectly.

See Also

Reference

View Node (listener)

View Node (object)

Method Resolution Criteria Dialog Box (Object Properties)

Use the **Method Resolution Criteria** dialog to provide detailed information about a specific method on the object view.

Use this	To do this
Method	View the name of the object's method for which the method resolution criteria are being set.
Endpoint	Select the endpoints on the local environment that are available for the type of resolution specified in **Type**. All endpoints in the LE are listed and available for selection. If an endpoint selected for one resolution type is already used with another resolution type then an error message will pop up when you click **Apply**. The error message explains which object and view has this endpoint used with another resolution type.
Method resolution criteria	View a description of the behavior for the IT runtime when it resolves a request to a method. The sequence of data flow events may require the TI to send and receive various amounts of data prior to obtaining the ability to make the connection between the incoming request and the target method.
Type	Select the type of resolution to be conducted: • **Endpoint** The endpoint resolution type is simplest means of resolving a request to a method. A data stream directed to a port always executes a single method on a view. This model is considered "raw sockets," and it represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Endpoint** disables **Object Name**, **Input format** and **Output format** and clears their contents. • **Transaction Request Message (TRM)** The TRM resolution type is specific to the IBM CICS Concurrent Server model and the Microsoft variant MSLink model. It represents a double exchange sequence. The first exchange represents the transaction request, and the second exchange represents the request and reply data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Transaction Request Message (TRM)** enables **Object**, **Input format**, and **Output format**. This option is available only for TCP/IP local environment/host environment pairs. • **Enhanced Listener Message (ELM)**: • **Data** Similar to **Endpoint** in that it adds the flexibility to identify a string in the data stream directed to a port that is used to associate the request with a specific method in the view. This model is considered "raw sockets." It represents a single synchronous exchange of data. When a new view is created and associated with the HE, the method resolves to the first endpoint defined in the associated local environment. Selecting **Data** disables **Object**, **Input format**, and **Output format** and clears their contents. • **Link to Program Name:** Resolves all the methods of a new object view using the data information specified when the object view was created. When a new view is created and associated with the HE, the method resolves to the Link Tran ID field in the default HE resolution. Selecting **Link to Program Name** disables **Object**, **Input format**, and **Output format** and clears their contents. This option is available only for SNA local environment/host environment pairs. Note Only **TRM-MSLink** and **Link to Program Name** are available for Link methods. Methods can be defined as Link in Visual Studio by setting the **Is Link** property to **True**.

Object Name	Select the object. The list of objects is populated from the Help string in the interface definition of objects identified to the resolution handler. Microsoft provides three TRM resolution handlers: • **Microsoft HIS - TRM Handler MS Link** (default) • **Microsoft HIS - TRM Handler MSCCS** • **Microsoft HIS - TRM Handler IBMCCS**
Input format	Select the format of the data stream that represents the TRM sent by the client host application program. The list of TRM input formats is determined by the UDT) definitions in the MicrosoftTRMDef.tim file in the installation TIMLibs directory. The **Input format** box is blank by default, but three input formats are defined that match the names of the resolution handler **Object**: • **MS Link Request Format** • **IBM Listener Type 1 Request Format** • **IBM Listener Type 2 Request Format**
Output format	Select the format of the data stream that represents the reply to the TRM that was sent by the host application program. TI formats a data stream that is returned to the host client application program in the form defined by the output TRM. The list of TRM output formats is determined by the UDT definitions in the MicrosoftTRMDef.tim file. The **Output format** box is blank by default, but three output formats are defined that match the names of the resolution handler **Object**: • **MS Link Reply Format** • **IBM Listener Type 1 Reply Format** • **IBM Listener Type 2 Reply Format**
Link	View the name of the program being linked to.
Program Name	Name of the TP program.
Resolution Data	View a definition of the data used by the IT runtime when it resolves a request to a method.
Data	Type the method resolution criteria for a method on the object view. The criteria can be a maximum of 256 alpha-numeric characters. This control is enabled for resolution types **Transaction Request Message (TRM)** and **Data**. This control is disabled for resolution type **Endpoint**.
Position	Type the resolution position. The position can be a maximum of nine numeric characters. This control is enabled for resolution type **Data** and disabled for resolution types **Transaction Request Message (TRM)** and **Endpoint**.

See Also

Reference

View Node (listener)

View Node (object)

General Tab (Remote Environment Properties)

Use the **General** tab to set the basic characteristics of the remote environment.

Use this	To do this
Type	View the type of remote environment, which describes the region on the mainframe with which your application is designed to work
Status	View the current state of the remote environment: • **Active**. The TI run-time environment accepts requests and attempts to communicate with the associated mainframe region. • **Inactive**. The TI run-time environment will not accept requests from client applications and returns an Inactive remote environment error message without attempting to communicate with the associated mainframe region. • **Disabled**.
Identifier	View the GUID for the remote environment. This identifier is useful when reviewing TI messages in the Windows Event Log.
Creator name	View the user ID of the user who created the remote environment.
Date created	View the date the remote environment was created.
Comment	Type or view a comment or description to help identify the new remote environment when its properties are displayed elsewhere in the product. You can enter up to 259 Unicode characters in this field. Useful information includes the name and location of the mainframe, and the name and telephone number of the system administrator.

⚠**Caution**
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

TCP/IP Tab (Remote Environment Properties)

Use the **TCP/IP** tab to define the network characteristics that the Windows environment uses to interact with the host.

Use this	To do this
IP/DNS Address	Type the address or select one from the drop-down list. The address can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used IP addresses.
TCP ports list	Type the TCP port number. The number can be a maximum of 256 positive numerals separated by semicolons. The port numbers cannot be repeated. The drop-down list displays the most recently used port numbers.
Time-out if no response within *number* seconds	Type the number of seconds. The send and receive time-out values are used by the WIP runtime environment when it communicates with the host environment. The time-out values are used on transport-specific APIs to terminate the receive API function when no host data or acknowledgement is received in the specified amount of time. The number of seconds can be a maximum of 3600 and a minimum of 0.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

CICS Tab (for TCP/IP Properties)

LU6.2 Tab (Remote Environment Properties)

Use the **LU 6.2** tab to define the network characteristics that the Windows environment uses to interact with the host.

Use this	To do this
Local LU alias	Select a local LU alias from the drop-down list, or type the alias to be used by this remote environment when contacting the mainframe. Create and configure this local LU alias through SNA Management (SNA Server)/SNA Manager (Host Integration Server) before you use the remote environment. The LU identified by the alias must correspond with a VTAM independent LU on the mainframe. The alias can be a maximum of 256 alphanumeric characters. The drop-down list displays all the local APPC LUs defined to the SNA Manager.
Remote LU alias	Select a remote LU alias from the drop-down list, or type the alias to be used by this remote environment when contacting the mainframe. Create and configure this remote LU alias through SNA Management before you use the remote environment. The partner LU identified by this alias must identify the ACBNAME of the CICS region, the partner LU associated between APPCMVS and IMS to reach IMS. The alias can be a maximum of 256 alphanumeric characters. The drop-down list displays all the remote APPC LUs defined to the SNA Manager.
Mode name	Select a mode name from the drop-down list, or enter the mode name to be used by this remote environment for sessions established with the mainframe. Create and configure this mode name through Host Integration Server SNA Manager before you use the remote environment. Ensure that a compatible mode has been defined on the mainframe. You can also obtain a list of configured modes from Host Integration Server **SNA Manager**. Note To use two-phase commit, the APPC mode must be Sync Level 2 capable.
Supports Sync Level 2 protocols	Select this option to enable Sync Level 2 protocols for this remote environment.
Time-out in seconds	View the time-out information used by the WIP runtime environment when it communicates with the host environment. The time-out values are used on transport-specific APIs to terminate the receive API function when no host data or acknowledgement is received in the specified amount of time. When a remote environment is first created, no timeout value is specified. The TI run-time environment will wait indefinitely for the mainframe transaction program to return output parameters and simultaneously block the calling client application until this response is received. This blocking behavior is typical for APPC applications. To avoid indefinite blocking, set a timeout value in the **Receive** box.
Receive	Type the number of seconds. The number of seconds can be a maximum of 3600 and a minimum of 0.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also
Reference
Remote Environments Node
Remote Environment Node
CICS Tab (for LU6.2 Link Properties)

Target Tab (Remote Environment Properties)

Use the **Target** tab to set the properties for the source of the captured data.

Use this	To do this
Use responses from the following remote environment	Select this option to specify a remote environment. A drop-down list appears containing all the remote environments that have the same network type, host environment, and programming model as the remote environment being defined. Responses returned from the target remote environment are stored in the recording file associated with this remote environment. If you choose this option, you must select a remote environment from the list box.
Remote environment list	Select the name of the remote environment that describes the mainframe from which you want to record responses.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

Recording Tab (Remote Environment Properties)

Use the **Recording** tab to set the properties for the file of captured data.

Use this	To do this
Recording selection	View information about the recording file in which the TI run-time environment stores responses for the components assigned to this remote environment.
Name	Type the name of the recording file. The file name can be a maximum of 256 alphanumeric characters. The drop-down list contains the most recently used recording files.
Browse	Launches the **Open File** dialog box for selecting a new or different recording file.
Recording information	View information about the recording.
Type	View the type of remote environment.
Last updated	View the data the remote environment was last updated.
Date created	View the date the remote environment was created.
Creator name	View the user ID of the user who created the remote environment.

⚠**Caution**
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

Locale Tab (Remote Environment Properties)

Use the **Locale** tab to define the computer data type characteristics used by the host that makes accepts requests from the Windows platform.

Use this	To do this
Locale	Select the language locale you want to use in selecting the code page used by the TI run-time e nvironment.
Use default code page for t he selected locale	Select this option to have TI select the default code page suitable for the specified locale. If you c lear this check box, you must explicitly select the code page.
Code page	Select the code page used to transform the incoming and outgoing data to a form that can be u sed by the host application program.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the pr operties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

Security Tab (Remote Environment Properties)

Use the **Security** tab to set the security properties of the remote environment for SNA.

Use this	To do this
Enable Single Sign-on	Select this option to require that the TI run-time environment uses Single Sign-on (SSO). Clear this option to allow TI to communicate with the mainframe region without setting user ID and password information.
Affiliate application	Select the SSO affiliate application to be queried for credentials. The list displays all the affiliate applications defined in the SSO. The display name of the affiliate application is a concatenation of the affiliate application name and description, separated by a hyphen.
COM+ Single Sign-on mapping use:	Identifies the source of the host user ID and password. Important This setting is for COM+ applications only. If you are using .NET applications, the credentials used by the TI runtime are the same credentials set on the IIS virtual directory where the .NET objects are stored. You can set or change the .NET credentials by using the "identity" element in the file web.config.
User credentials	Select this option to have the TI run-time environment obtain the user ID and password information associated with the user logon in which the client application is executing. Caution For the COM server to impersonate the security credentials of the client application, the client application must grant the COM server permission to impersonate it.
COM+ application credentials	Select this option to have the TI run-time environment obtain the user ID and password information as defined for a component's COM+ application. To administer this security information, use Component Services (COM+) in Windows Server 2003. This setting is ignored when a TI component is deployed in a COM+ application which itself is configured as a Library application (instead of a Server application) on the Activation tab of the COM+ application property sheet. A Library application runs using the Windows user credentials of the process that invoked it, and TI attempts to map this name to a corresponding host user name.
Allow application to override selected authentication	Select this option to enable the TI run-time environment to use the application override mechanism for obtaining user ID and password information. The client application must be written to support the TI security override mechanism. Clear this option to base the user ID and password solely on previously selected user or package credentials. This security option can be applied to a TI component deployed in a Library COM+ or ASP.NET application or in a Server COM+ or ASP.NET application. This option appears only if **Enable Single Sign-on** is selected.
Require client provided security	Select this option to require the client to provide the security credentials, either by using the client context or by making call-back security available. This option appears only if **Enable Single Sign-on** is cleared.
Use already verified or persistent verification authentication	Select this option to have the TI run-time environment use "Already Verified," "Persistent Verification" or "No authentication," depending on the configuration of the mainframe region. "Already Verified" sends only the user ID and no password. "Persistent Verification" sends a user ID and password on the first transaction from the given user and only the user ID thereafter. "No authentication" neither sends a user ID nor a password. The host configuration controls the actual authentication to be used when this option is enabled. In a CICS region, for example, these settings correspond to the ATTACHSEC options Identify, Persistent, and None, respectively. This security option can be applied to a TI component deployed in a Library application or in a Server application. This security option can be applied to a TI component deployed in a Library COM+ application or in a Server COM+ application.

⚠**Caution**
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

Concepts

How To Impersonate Client Application Security Credentials

Security Tab (Remote Environment Properties)

Use the **Security** tab to set the security properties of the remote environment for TCP/IP.

Use this	To do this
Enable Single Sign-on	Select this option to require that the TI run-time environment uses Single Sign-on (SSO). Clear this option to allow TI to communicate with the mainframe region without setting user ID and password information.
Affiliate application	Select the SSO affiliate application to be queried for credentials. The list displays all the affiliate applications defined in the SSO. The display name of the affiliate application is a concatenation of the affiliate application name and description, separated by a hyphen.
COM+ Single Sign-on mapping use:	Identifies the source of the host user ID and password. Important This setting is for COM+ applications only. If you are using .NET applications, the credentials used by the TI runtime are the same credentials set on the IIS virtual directory where the .NET objects are stored. You can set or change the .NET credentials by using the Identity tag in the file web.config.
User credentials	Select this option to have the TI run-time environment obtain the user ID and password information associated with the user logon in which the client application is executing. Note This setting is ignored when a TI component is deployed in a COM+ application which itself is configured as a Library application (instead of a Server application) on the Activation tab of the COM+ application property sheet. A Library application runs using the Windows user credentials of the process that invoked it, and TI attempts to map this name to a corresponding host user name. Caution For the COM server to impersonate the security credentials of the client application, the client application must grant the COM server permission to impersonate it.
COM+ application credentials	Select this option to have the TI run-time environment obtain the user ID and password information as defined for a component's COM+ application. To administer this security information, use Component Services (COM+) in Windows Server 2003. This setting is ignored when a TI component is deployed in a COM+ application which itself is configured as a Library application (instead of a Server application) on the Activation tab of the COM+ application property sheet. A Library application runs using the Windows user credentials of the process that invoked it, and TI attempts to map this name to a corresponding host user name.
Allow application to override selected authentication	Select this option to enable the TI run-time environment to use the application override mechanism for obtaining user ID and password information. The client application must be written to support the TI security override mechanism. Clear this option to base the user ID and password solely on previously selected user or package credentials. This security option can be applied to a TI component deployed in a Library COM+ or ASP.NET application or in a Server COM+ or ASP.NET application. Note This option appears only if **Enable Single Sign-on** is selected.
Require client provided security	Select this option to require the client to provide the security credentials, either by using the client context or by making call-back security available. Note This option appears only if **Enable Single Sign-on** is cleared.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

Concepts

How To Impersonate Client Application Security Credentials

TCP/IP Tab (Remote Environment Properties)

Use the TCP/IP tab to set the IP address and port number for the host environment.

Use this	To do this
IP/DNS address	Type the address or select from the drop-down list. The address can be a maximum of 256 alphanumeric characters. The drop-down list displays the most recently used IP addresses.
Port list	Type the TCP port number. The number can be a maximum of 256 positive numerals separated by semicolons. The port numbers cannot be repeated. The drop-down list displays the most recently used port numbers.
Time-out in seconds	The **Send** and **Receive** time-out values are used by the TI runtime environment when it communicates with the host environment. The time-out values are used on transport-specific application program interfaces (APIs) to terminate the send and/or receive API functions if no host acknowledgement is received in the specified amount of time.
Receive	Type the number of seconds the host should wait for a response before it times out. The maximum number of seconds is 3,600; the default is **30** seconds.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

CICS Tab (for LU6.2 Link Properties)

Use the **CICS** tab to set host-specific CICS properties.

Use this	To do this
Transaction name	Type or select the transaction name on CICS tied to CICS program DFHMIRS.
Allow use of explicit SYNCPOINT commands for non-transactional components	Select this option to allow a CICS transaction program that is accessed as a nontransactional component to call EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK commands. Clear this option to make any CICS transactional component that calls SYNCPOINT commands fail with an ADPL CICS ABEND. LU 6.2 Resync services must be configured on the SNA Local LU. Enable Sync Point must be enabled on SNA Remote LU. The Mode Table associated with the call must have sync level 2 enabled. The typelib must have transaction support properly set. The AppInt .Net model doesn't support Transactional nor does AppInt Com over TCP/IP.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

LU6.2 Tab (Remote Environment Properties)

CICS Tab (for TCP/IP Properties)

Use the **CICS** tab to set host-specific CICS properties.

Use this	To do this
Use IBM supplied CICS TCP/IP exit routine	When security is enabled, the default TCP/IP header contains the userid followed by the password. Selecting **Use IBM supplied CICS TCP/IP exit routine**,however, overrides the default behavior and causes the TCP/IP header to put the password before the userid.

⚠**Caution**
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

TCP/IP Tab (Remote Environment Properties)

IMS Tab

Use the IMS tab to identify the IBM-defined exit program on the host.

Use this	To do this
ITOC information	View information about the ITOC exit name.
HWSIMSO0	Select this option to use IMS Connect with exit HWSIMSO0.
HWSIMSO1	Select this option to use IMS Connect with exit HWSIMSO1.
IMS system ID	Type the IMS system ID or select from the drop-down list. The ID can be a maximum of 8 alphanumeric characters. The drop-down list displays the most recently used IMS system IDs.
ITOC exit name	Type the ITOC exit name or select from the drop-down list. The name can be a maximum of 8 alphanumeric characters. The drop-down list displays the most recently used ITOC exit names.

Caution
The properties of a remote environment are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the remote environment to function incorrectly.

See Also

Reference

Remote Environments Node

Remote Environment Node

SSO tab (Security Policy Properties)

Use the SSO tab to set the security properties of the remote environment.

Use this	To do this
SSO affiliate application	Select the SSO affiliate application to be queried for credentials. The list displays all the affiliate applications defined in the SSO. The display name of the affiliate application is a concatenation of the affiliate application name and description, separated by a hyphen.

Caution
The properties of a security policy are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the security policy to function incorrectly.

See Also

Reference

Security Policies Node

Security Policy Node

General Tab (Object Properties)

Use the **General** tab to define the basic characteristics of the object that will service requests from the Windows platform.

Use this	To do this
Object	View the ProgID that names the component. This ProgID is created by Visual Studio in Host Integration Server (Designer in SNA Server) using the type library name, the interface name, and the version information given for the component.
Type	View the type of remote environment for which this component is designed. This type is specified in Visual Studio when the component's type library is created. Note You cannot change the type of an object.
Description	View the component description entered in Visual Studio when the component's type library was created.
File	View the name of the component library.
CLSID	View the CLSID that uniquely identifies the component.
Comment	Type a comment that provides additional information about the use of the object. The comment can be a maximum of 259 Unicode characters.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (WIP)

Object Node (WIP)

Hosting Tab (COM Object Properties)

Use the **Hosting** tab to set the hosting properties of the object.

Use this	To do this
COM+ application	Select a COM application. This list is populated by enumerating the COM+ applications on the local computer.
Virtual directory	Select an IIS virtual directory. This list is populated by enumerating the IIS Virtual Directories on the local computer.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (WIP)

Object Node (WIP)

Hosting Tab (.NET Object Properties)

The **Hosting** tab allows you to set the hosting properties of the object.

Use this	To do this
.NET assembly	Select a .NET assembly. This list is populated by enumerating the .NET assemblies on the local computer.
Virtual directory	Select an IIS virtual directory. The list is populated from enumerating the IIS Virtual Directories on the local computer.
Key file for proxy	Select the key (.snk) file that TI Manager uses to sign the proxy in the IIS virtual directory. The list is populated with files used previously.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (WIP)

Object Node (WIP)

Remote Environment Tab (Object Properties)

Use the **Remote Environment** tab to set the remote environment properties of the object.

Use this	To do this
Remote environment	View the name of the remote environment to which this object is assigned. This list is populated by the remote environments already defined and that agree in network type, host environment and programming model. You can select a different remote environment for this object.

Caution
The properties on an object are not intended to be set or changed programmatically. Setting or changing the properties programmatically might cause the object to function incorrectly.

See Also

Reference

Objects Node (WIP)

Object Node (WIP)

Enterprise Single Sign-On Help

This section provides instructions for the Enterprise Single Sign-On (SSO) user interface (UI) for Host Integration Server. You access this information by using the F1 key or by clicking **Help** in the UI.

In This Section

Affiliate Applications Properties

Use these property pages to manage your Affiliate Applications.

This section contains:

Affiliate Applications

Use these menu commands to configure and manage Affiliate Applications.

Affiliate Applications Properties: Accounts

Use this dialog box to view or modify access accounts for the Affiliate Application.

Use this	To do this
Application Administrators	The Windows account that will manage this Affiliate Application.
Application Users	The Windows account that will be used to access this Affiliate Application.

See Also

Other Resources

Enterprise Single Sign-On Help

Affiliate Applications Properties: Fields

Use this dialog box to view or change fields for the Affiliate Application.

Use this	To do this
Number of fields	Determines the number and type of credentials (user ID, password, smart card) that users must provide to connect to the affiliate application. You can enter as many fields as there are credentials for the affiliate application, but the first field must be the user ID.
Masked	Determines whether this credential is masked (that is, whether the characters that the user types are displayed on the screen). You cannot change this property after you create the application.
Synchronized	Determines whether this field is synchronized (used with password synchronization). Only one field can be marked as synchronized. You cannot change this property after you create the application.

See Also

Other Resources

Enterprise Single Sign-On Help

Affiliate Applications Properties: General

Use this dialog box to view or change general properties for the affiliate application.

Use this	To do this
Application type	Enterprise Single Sign-On (SSO) defines several different application types. The different application types support different types of mappings between the Windows account and the account on the non-Windows system. The application types are: **Individual**: Individual applications support one-to-one mappings between the Windows account and the non-Windows account. In an Individual type application, one Windows account is mapped to one, and only one, non-Windows account. The mapping can be used in either direction, from Windows to non-Windows, or from non-Windows to Windows, or both, depending on the flags that have been set for this application. Thus, Individual applications may be used for Windows initiated SSO, Host initiated SSO, or both. **Group**: Group mapping consists of mapping a Windows group, which contains multiple Windows users, to a single account in the affiliate application. You can also specify multiple accounts for the SSO Application Users role. Each account that you specify can be associated with an external account. For example, it is possible to map a domain user (domain\userA) to one set of external credentials (ExternalUser1). The same domain\userA could also be a member of Domain Group (domain\group1) and this group could be mapped to a different set of external credentials (ExternalUser2). In this case, it is important that the administrator (who must be an Application Administrator or above) specifies the correct order for the Application Users accounts. The mapping for the first account (in the Order) of which the caller is a member is the one that will be used. In this case, if mapping for domain\userA to ExternalUser1 is set to Order 0, SSO will return this set of credentials for domain\UserA. Only an application administrator, SSO affiliate administrator, or SSO administrator can create a group mapping. You cannot specify the same group application for Windows-initiated SSO and Host-initiated SSO. **Host Group**: Host Group applications are conceptually the reverse of Group applications. They support mappings between a defined group of non-Windows accounts to a single Windows account. The single Windows account that will be used by the non-Windows accounts is defined by the Application Users account for the application. The group of non-Windows accounts that is allowed to access this application is defined by creating a mapping for each non-Windows account. Host Group applications may only be used for Host initiated SSO. Important Mappings can be created only for Windows domain accounts. Local accounts cannot be mapped. Important When you use group mappings, the members of the group can obtain the credential information for the group mapping.
Application name	Name of the affiliate application. You cannot change this property after you create the affiliate application.
Description	Brief description of the affiliate application.
Contact information	The main contact for this affiliate application that users can use. (Can be an e-mail address.)
Allow local accounts for access accounts	Determines whether you allow the use of local groups and accounts in the SSO system. You should select this option only in single-computer scenarios.

Use SSO Affiliate Admin Accounts for Application Admin accounts	Determines whether the Application Admin accounts will be the same as the SSO Affiliate Admin accounts. Default is not selected.

See Also

Other Resources

Enterprise Single Sign-On Help

Affiliate Applications Properties: Options

Use this dialog box to view or change options for the Affiliate Application.

Use this	To do this
Enabled	The status (enabled/disabled) of this affiliate application.
Allow Windows initiated SSO	Select if Windows initiated Enterprise Single Sign-On (SSO) is allowed.
Disable credential cache	The SSO Server stores credentials in a cache to expedite access. Default is not checked.
Tickets allowed	Determines whether the SSO system uses tickets for this affiliate application. You must be an SSO administrator to select this option.
Validate tickets	Determines whether the SSO system validates tickets when the user redeems them. You must be an SSO administrator to select this option.
Timeout tickets	Determines whether tickets have an expiration time. Default is checked. Unless it is required, do not disable ticket timeouts. You must be an SSO administrator to set this option.
Ticket timeout (in minutes)	Specifies a ticket timeout specific to the affiliate application. This can be set only when updating an affiliate application, not when creating it. If ticketing is enabled for this application and this property is not, the timeout specified at the SSO System (Global) level is used. You must be an SSO administrator to set this option.
Allow host initiated SSO	Select this if it is a host initiated SSO type application. Default is not checked.
Verify external credentials	Select to verify external credentials.

See Also
Other Resources
Enterprise Single Sign-On Help

Create New Affiliate Application Wizard

Use this wizard to create a new Affiliate Application.

This section contains:

Create New Affiliate Application Wizard: Welcome

Click **Next** to continue.

Create New Affiliate Application Wizard: Accounts

Specify the access accounts for the new Affiliate Application. You can specify one or more accounts for both Application Administrators and Application Users.

Note
In certain Workgroup environments, clicking the **Browse** button will cause this screen to close. Instead of using the **Browse** button, simply type the name of the required accounts in the box.

For group type Affiliate Applications, it is important to note that when specifying multiple accounts for Application Users, the way in which the accounts are ordered is very important. This is because different ordering of accounts can result in different credentials being returned, which could potentially result in an authentication failure.

For example, UserA could be a member of two groups: Group1 and Group2. Each of these groups could in turn be mapped to an account as follows:

- Group1 is mapped to ExternalCredentials1
- Group2 is mapped to ExternalCredentials2

If the order specified for 'Application Users' is Group1;Group2, then when the credentials are requested for UserA, SSO returns ExternalCredentials1.

However, if the order specified for 'Application Users' is Group2;Group1, then SSO returns ExternalCredentials2.

Application Administrators

The Windows account(s) that will manage this Affiliate Application.

Application Users

The Windows account(s) for which mappings can be created.

See Also
Other Resources
Enterprise Single Sign-On Help

Create New Affiliate Application Wizard: Fields

Specify fields for the new Affiliate Application.

Use this	To do this
Number of fields	Determines the number of fields that users must provide to connect to the affiliate application. You can enter as many fields as there are credentials for the affiliate application, but the first fiel d must be the external user ID.
Masked	Determines whether this field is masked (that is, whether the characters that the user types are d isplayed on the screen). You cannot change this property after you create the application.
Synchronized	Determines whether this field is synchronized (used with password synchronization). Only one fi eld can be marked as synchronized. You cannot change this property after you create the application.
Credentials are Windows c redentials	Increases security by identifying credentials as Windows credentials; applies to individual applica tions only.
Credentials are restricted	Applies only to group applications. This option is only necessary when using Enterprise Single Si gn-On with Microsoft Office 2007.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Affiliate Application Wizard: General

Specify general properties for the new Affiliate Application.

Use this	To do this
Application type	Enterprise Single Sign-On (SSO) defines several different application types. The different application types support different types of mappings between the Windows account and the account on the non-Windows system. The application types are: **Individual**: Individual applications support one-to-one mappings between the Windows account and the non-Windows account. In an Individual type application, one Windows account is mapped to one, and only one, non-Windows account. The mapping can be used in either direction, from Windows to non-Windows, or from non-Windows to Windows, or both, depending on the flags that have been set for this application. Thus, Individual applications may be used for Windows initiated SSO, Host initiated SSO, or both. **Group**: Group applications support mappings between one or more Windows groups or users to one single non-Windows account. The Application Users account is used to define the Windows groups or users that will be used for this Group application. **Host Group**: Host Group applications are conceptually the reverse of Group applications. They support mappings between a defined group of non-Windows accounts to a single Windows account. The single Windows account that will be used by the non-Windows accounts is defined by the Application Users account for the application. The group of non-Windows accounts that is allowed to access this application is defined by creating a mapping for each non-Windows account. Host Group applications may only be used for Host initiated SSO.
Application name	Name of the affiliate application. You cannot change this property after you create the affiliate application.
Description	Brief description of the affiliate application.
Contact information	The main contact for this affiliate application that users can use. (Can be an e-mail address.)
Allow local accounts for access accounts	Determines whether you allow the use of local groups and accounts in the SSO system. You should only select this option in single-computer scenarios.
Use SSO Affiliate Admin Accounts for Application Admin accounts	Determines whether the Application Admin Accounts will be the same as the SSO Affiliate Admin Accounts. Default is not selected.

See Also
Other Resources
Enterprise Single Sign-On Help

Create New Affiliate Application Wizard: Options

Specify options for the new Affiliate Application.

Use this	To do this
Enabled	The status (enabled/disabled) of this affiliate application.
Allow Windows initiated SSO	Select if Windows initiated Enterprise Single Sign-On (SSO) is allowed. Default is selected.
Disable credential cache	The SSO server stores credentials in a cache to expedite access. Default is not selected.
Tickets allowed	Determines whether the SSO system uses tickets for this affiliate application. You must be an SSO administrator to select this option.
Validate tickets	Determines whether the SSO system validates tickets when the user redeems them. You must be an SSO Administrator to select this option.
Timeout tickets	Determines whether tickets have an expiration time. Default is checked. Unless it is required, do not disable ticket time-outs. You must be an SSO Administrator to set this option.
Ticket timeout (in minutes)	Specifies a ticket time-out specific to the affiliate application. If ticketing is enabled for this application and this property is not, the time-out specified at the SSO System (Global) level is used. You must be an SSO Administrator to set this option.
Allow host initiated SSO	Select this if Host initiated SSO is allowed. Default is not selected.
Verify external credentials	Select to verify external credentials.
Direct Password Sync from Windows	Select to allow Direct Password Sync for this application.
Application users cannot create mappings	Increases system security by allowing only administrators to create mappings.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Affiliate Application Wizard: Finish

Click **Finish** to close the Wizard.

Create New Password Sync Adapter Wizard

Use this wizard to create a new Password Sync Adapter.

This section contains:

Create New Password Sync Adapter Wizard: Welcome

Click **Next** to continue.

Create New Password Sync Adapter Wizard: Accounts

Specify access accounts for the new Password Sync Adapter.

Use this	To do this
Application Administrators	The Windows account that will manage this adapter.
Application Users	The Windows account that will be used to access this adapter.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Password Sync Adapter Wizard: General

Specify general properties for the new Password Sync Adapter.

Use this	To do this
Adapter name	Name of the adapter.
Description	Brief description of the adapter.
Computer	Name of the computer on which this adapter will run. Must be the fully qualified computer name.
Group adapter	Select if this adapter is a group adapter.
Allow local accounts for access accounts	Determines whether the App Admin or App Users accounts can be local account s. Default is not selected.
Use SSO Affiliate Admin Accounts for Ap plication Admin Accounts	Determines whether the Application Admin Accounts will be the same as the SS O Affiliate Admin Accounts. Default is not selected.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Password Sync Adapter Wizard: Options

Specify options for the new Password Sync Adapter.

Use this	To do this
Enabled	Determines whether the adapter is enabled.
Receive password changes from adapter	Determines whether the adapter is allowed to receive external password changes. Default is not selected.
Verify old password	Determines whether the adapter will verify the old password when an external password change is received. If this option is selected then with an external password change the external adapter must supply the old external password as well as the new external password. The old external password is then compared with the existing external password in the Enterprise Single Sign-On (SSO) database for that external account. If they match, the password change is accepted. If they do not match, the password change is rejected. Default is selected.
Change Windows password	Determines whether the Windows password in Active Directory will also be changed when an external password change is received (full sync). Default is not selected.
Send Windows password changes to adapter	Determines whether Windows password changes made in Active Directory will be sent to the external adapter. Default is not selected.
Send old password to adapter	If selected, the old password value (from the SSO database) will also be sent to the external adapter as well as the new password value. Some external systems might require both the old and new password values to change the password. Default is not selected.
Allow mapping conflicts	Determines whether the adapter will allow mapping conflicts. A mapping conflict occurs when mappings are not unique. In a single SSO Individual application, mappings are always one-to-one: one Windows account is mapped to exactly one external account and vice versa. However, it is possible to assign more than one application to an adapter. Thus, it is possible to have a mapping in one application that conflicts with a mapping in the other. This purpose of this option is to prevent this from occurring. It is more secure to not allow mapping conflicts unless there is a specific, well understood requirement for this behavior. Default is not selected.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Password Sync Adapter Wizard: Properties

Specify properties for the new Password Sync Adapter.

Use this	To do this
Properties file	The location of the file containing the adapter properties.
Notification Retry Count	Default is 5.
Notification Retry Delay (in mins)	Default is 1.
Maximum Pending Notifications	Default is 8.
Store Notifications (when offline)	Check to store notifications in a local replay file when the database cannot be contacted.

See Also

Other Resources

Enterprise Single Sign-On Help

Create New Password Sync Adapter: Finish

Click **Finish** to close the Wizard.

Mapping Wizard

Use this wizard to create and configure mappings.

In This Section

Create New Mappings Wizard: Welcome

Use this page to enter basic information about the Affiliate Application.

Affiliate Application name

The name of the application to be mapped.

Description

A description (optional).

Create New Mappings Wizard: Mappings File Option

Use this page to determine whether a new or existing mappings file will be used.

Generate new mappings file

Select to generate a new file. Default is selected.

Use an existing mappings file

Select to use an existing file.

Create New Mappings Wizard: Files Location

Mappings file

If necessary, browse to a new file.

Log file

Use the default, or change if necessary.

Validate

Click to ensure that you have proper access and have not entered invalid information.

Create New Mappings Wizard: Accounts

Application Users Accounts

Select the appropriate accounts.

Validate

Click to see the results in the status window.

Create New Mappings Wizard: External User Name

This page collects information to generate an external account name based on the Windows account name.

Windows domain name using

Change or accept these domain name properties as appropriate.

Windows user name using

Change or accept these user name properties as appropriate.

Remove characters

Remove characters as necessary.

Prepend characters

Add characters to the beginning of the name.

Domain/user separator characters

Select separator characters.

Append characters

Add characters to the end of the name.

Limit the number of characters

Limit the number of characters for the name.

Example

Uses the data entered above to display the Windows domain name, Windows user name, and external user name.

Create New Mappings Wizard: Generate

Mappings file location

Displays the location entered previously.

Selected accounts

Displays the accounts specified previously.

Start

Click to generate mappings.

Stop

Click to pause mappings.

View log file

Click to open the mappings log file in Notepad.

Create New Mappings Wizard: Options

Mappings file location

Displays the location you entered previously.

View/Edit mappings file

Click to open the file for editing as necessary.

Enable mappings

Select to enable mappings after they are created. Default is selected.

Set password

Selecting this option allows you to choose one of the following:

- Same as external name
- Define from Windows user account
- Fixed value (enter and confirm a new password)

Create New Mappings Wizard: Password

This page collects information to generate a password based on the Windows account name.

Windows domain name using

Change or accept these domain name properties as appropriate.

Windows user name using

Change or accept these user name properties as appropriate.

Remove characters

Remove characters as necessary.

Prepend characters

Add characters to the beginning of the name.

Domain/user separator characters

Select separator characters.

Append characters

Add characters to the end of the name.

Limit the number of characters

Limit the number of characters for the name.

Example

Uses the data entered above to display the Windows domain name, Windows user name, and password.

Create New Mappings Wizard: Create

Use this page to create the mappings file with all of the information that has been collected.

Mappings file location

Confirm location of the file.

View mappings file

Click to view as necessary.

Start

Click to generate the mappings file.

Stop

Click to pause.

Create New Mappings Wizard: Finish

View Log File

Click to show the log file generated in the mapping process.

When I click finish, delete the mappings file

Select if desired. Default is selected unless you used an existing mappings file.

When I click finish, delete the log file

Select if desired. Default is selected unless you used an existing file.

Done

Click to close the wizard and display the results of the mapping process.

Enterprise Single Sign-On System

Use these topics to view and change the Enterprise Single Sign-On system properties.

This section contains:

Enterprise Single Sign-On

Use these menu commands to configure and run Enterprise Single Sign-On.

SSO System Properties: Accounts

This screen displays account properties for the overall Enterprise Single Sign-On (SSO) system.

Use this	To do this
SSO Administrators	The Windows account that will manage this SSO system.
SSO Affiliate Administrators	The Windows accounts that define who can create and manage all Affiliate Applications.

See Also

Other Resources

Enterprise Single Sign-On Help

SSO System Properties: Audits

These properties define the sizes of the audit tables in the Enterprise Single Sign-On (SSO) database.

Use this	To do this
Deleted applications	Default is 1,000.
Deleted mappings	Default is 1,000.
External credential lookups	Default is 1,000.

See Also

Other Resources

Enterprise Single Sign-On Help

SSO System Properties: General

This screen displays general properties for the overall SSO system.

Use this	To do this
Master secret server	The master secret server is the Enterprise Single Sign-On (SSO) server that stores the master secret (encryption key). The master secret server generates the master secret when an SSO Administrator requests it. The master secret server stores the encrypted master secret in the registry. Only SSO Administrators can access the master secret.
Ticket timeout (in minutes)	This property specifies the length of time for which a ticket that SSO issues is valid. To satisfy most of the scenarios in an enterprise that use SSO, the default ticket time-out is 2 minutes. The SSO Administrator can change this based on the application requirements.
Credential cache timeout (in minutes)	This property specifies the credential cache time-out for all SSO servers. SSO servers cache the credentials after the first lookup. By default, the credential cache time-out is 60 minutes. The SSO Administrator can change this to a suitable value based on the security requirements.

See Also

Other Resources

Enterprise Single Sign-On Help

SSO System Properties: Options

This screen displays options for the overall Enterprise Single Sign-On (SSO) system.

Use this	To do this
Allow tickets	Determines whether tickets are allowed at the system level. Only an SSO Administrator can configure tickets at the SSO system level and at the Affiliate Application level. If ticketing is disabled at the system level, it cannot be used at the Affiliate Application level either. It is possible to enable tickets at the system level and disable them at the Affiliate Application level.
Allow host initiated SSO	By default, host initiated Single Sign-On is not enabled in the Single Sign-On system, and must be enabled by the SSO Administrator.
From Windows to adapters	Determines whether Windows password changes (in Active Directory) will be sent to password sync adapters. Default is not selected.
From adapters to SSO database (partial sync)	Determines whether external password changes received from password sync adapters will cause the password to be changed in the SSO database. Default is not selected.
From adapters to Windows (full sync)	Determines whether external password changes received from password sync adapters will cause the Windows password to be changed in Active Directory. Default is not selected.

See Also

Other Resources

Enterprise Single Sign-On Help

System

Use these menu commands to configure and run your Enterprise Single Sign-On system.

System Main

The information on this page applies to the overall Enterprise Single Sign-On (SSO) system.

Use this	To do this
SSO Server	Name of the SSO server that is currently being used by the snap-in.
SQL Server	Name of the SQL server that the SSO server is using.
SSO database	Name of the SSO database that the SSO server is using.
SSO secret server	Name of the SSO master secret server.
SSO status	Status of the SSO system (enabled/disabled).
SSO Administrators	The accounts containing the Administrators for the SSO system.
SSO Affiliate Administrators	The accounts containing the Affiliate Administrators for the SSO system.

See Also

Other Resources

Enterprise Single Sign-On Help

System Main Menu

Use the commands on this menu to manage the overall ESSO system.

Use this	To do this
Create database	Create a new database.
Upgrade Database	Upgrade the current database.
Generate Secret	Generate the Master Secret.
Back up Secret	Back up the Master Secret.
Restore Secret	Restore the Master Secret.

See Also

Other Resources

Enterprise Single Sign-On Help

Password Sync Adapter Properties

Use these property pages to configure Password Sync Adapters.

This section contains:

Password Sync Adapter Properties: Accounts

Use this dialog box to view or change the access accounts for the Password Sync Adapter.

Use this	To do this
Application Administrators	The Windows account that will manage this adapter.
Application Users	The Windows account that will be used to access this adapter.

See Also

Other Resources

Enterprise Single Sign-On Help

Password Sync Adapter Properties: General

Use this dialog box to view or change the general properties for the Password Sync Adapter.

Use this	To do this
Adapter name	Name of the adapter.
Description	Brief description of the adapter.
Computer	Name of the computer on which the adapter will run. Must be the fully qualified computer name.
Group adapter	Will be selected if this adapter is a group adapter.
Allow local accounts for access accounts	Determines whether the App Admin or App Users accounts can be local accounts. Default is not selected.
Use SSO Affiliate Admin Accounts for Application Admin Accounts	Determines whether the Application Admin Accounts will be the same as the Enterprise Single Sign-On (SSO) Affiliate Admin Accounts. Default is not selected.

See Also

Other Resources

Enterprise Single Sign-On Help

Password Sync Adapter Properties: Options

Use this dialog box to view or change options for the Password Sync Adapter.

Use this	To do this
Enabled	Determines whether the adapter is enabled.
Receive password changes from adapter	Determines whether the adapter is allowed to receive external password changes. Default is not selected.
Verify old password	Determines whether the adapter will verify the old password when an external password change is received. If this option is selected then with an external password change the external adapter must supply the old external password as well as the new external password. The old external password is then compared with the existing external password in the Enterprise Single Sign-On (SSO) database for that external account. If they match, the password change is accepted. If they do not match, the password change is rejected. Default is selected.
Change Windows password	Determines whether the Windows password will also be changed in Active Directory when an external password change is received (full sync). Default is not selected.
Send Windows password changes to adapter	Determines whether Windows password changes will be sent to the external adapter. Default is not selected.
Send old password to adapter	If selected, the old password value (from the SSO database) will also be sent to the external adapter as well as the new password value. Some external systems might require both the old and new password values to change the password. Default is not selected.
Allow mapping conflicts	Determines whether the adapter will allow mapping conflicts. A mapping conflict occurs when mappings are not unique. In a single SSO Individual application, mappings are always one-to-one: one Windows account is mapped to exactly one external account and vice versa. However, it is possible to assign more than one application to an adapter. Thus, it is possible to have a mapping in one application that conflicts with a mapping in the other. This purpose of this option is to prevent this from occurring. It is more secure to not allow mapping conflicts unless there is a specific, well understood requirement for this behavior. Default is not selected.

See Also
Other Resources
Enterprise Single Sign-On Help

Password Sync Adapter Properties: Properties

Use this dialog box to view or change properties for the Password Sync Adapter.

Use this	To do this
Properties file	The location of the file containing the adapter properties.
Notification Retry Count	Default is 5.
Notification Retry Delay (in mins)	Default is 1.
Maximum Pending Notifications	Default is 8.
Store Notifications (when offline)	Select to store notifications in a local replay file when the database cannot be contacted.

See Also

Other Resources

Enterprise Single Sign-On Help

Password Synchronization

Use these menu commands to configure and run Password Synchronization.

Password Sync Adapter Properties: System

Use this dialog box to view or change properties for the Password Sync Adapter.

Notification Retry Count

Default is 5.

Notification Retry Delay (in mins)

Default is 1.

Maximum Pending Notifications

Default is 8.

Store Notifications (when offline)

Select to store notifications in a local replay file when the database cannot be contacted.

Password Sync Adapter Properties: Custom

Use this screen to view or change the Properties file information for the Password Sync Adapter.

Properties file

The location of the file containing the adapter properties.

Create Filter Wizard

Use this wizard to create and configure Password Filters.

In This Section

Create Filter Wizard: Welcome

Click **Next** to continue.

Create Filter Wizard: General

Use this page to enter basic information.

Name

Enter a name.

Description

Enter a description

Create Filter Wizard: Basic

Set these properties according to any restrictions on your host system.

Format

Select upper case, lower case, or as is.

Remove these characters

Specify characters not supported on the host end.

Maximum length

Specify as needed.

Clear

Clears the options.

Filter results

Displays filtered results compared to original.

Create Filter Wizard: Advanced

Set these properties according to any restrictions on your host system.

Substitute

If the host system does not recognize certain characters, use this control to specify replacement characters.

Padding

If padding is required, specify for padding to be placed at the beginning or end of the password.

Minimum length

Set a minimum length.

Pad character

Enter a character to be used for padding.

Clear

Clears the options.

Filter results

Displays filtered results compared to the original.

Create Filter Wizard: Finish

Click **Finish** to close the wizard.

Filter Properties: Basic

Set these properties according to any restrictions on your host system.

Format

Select upper case, lower case, or as is.

Remove these characters

Specify characters not supported on the host end.

Maximum length

Specify as needed.

Clear

Clears the options.

Filter results

Displays filtered results compared to original.

Filter Properties: Advanced

Set these properties according to any restrictions on your host system.

Substitute

If the host system does not recognize certain characters, use this control to specify replacement characters.

Padding

If padding is required, specify for padding to be placed at the beginning or end of the password.

Minimum length

Set a minimum length.

Pad character

Enter a character to be used for padding.

Clear

Clears the options.

Filter results

Displays filtered results compared to the original.

Server Properties

Use these topics to view and configure Server properties.

In This Section

Server Properties: Audit Levels

These properties reflect the server status as of the most recent refresh.

Positive Audit Level

Select High, Low, or Medium.

Negative Audit Level

Select High, Low, or Medium.

Status

Displays the online or offline status of the server.

Server Properties: SSO Database

This screen displays information about the SSO Database.

SQL Server

Name of the SQL Server.

SSO Database

Name of the SSO Database.

Use SSL when connecting to SQL Server

Select this to increase security.

Server Properties: SSO Service

Use this page to enter information for the SSO Service.

Service Account

Enter the name of the SSO Service Account.

Password

Enter a password.

Confirm password

Re-enter the password.

Server Properties: Password Sync Properties

Password Sync Age (in hours)

Specify the desired age.

Allow Password Sync from PCNS

Select if desired.

Allow Password Sync from MIIS

Select if desired.

Server Properties: Advanced

Allow Remote Lookup

Select to enable lookups from a remote computer.

Data Integration Help

Use the topics in this section to navigate through the Data Integration User Interface. Click a topic below for more information.

In This Section

- Data Source Wizard
- Data User Interface Elements
- Configuring a Data Source

Data Source Wizard

The Data Source Wizard guides you through the configuration process. The wizard dynamically adapts to both DB2 and VSAM data sources and displays the appropriate screens.

In This Section

Welcome Screen

Click **Next** to continue.

Data Source Screen

Data source platform

Select the appropriate platform from the dropdown list.

Network type

Select either TCP/IP or SNA LU 6.2 (APPC).

TCP/IP Network Connection Screen

Address or alias

When TCP/IP Connection is selected as the network transport, this field indicates the IP address of the host DB2 server.

Port

When TCP/IP Connection is selected as the network port, this field indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default is **IP port 446**.

Distributed transactions

When this option is checked, two-phase commit (distributed unit of work) is enabled. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the Host Integration Server TCP/IP Resync service.

APPC Network Connection Screen

Local LU alias

When APPC Connection is selected, this field is the name of the local LU alias configured in Host Integration Server.

Remote LU alias

When APPC Connection is selected, this field is the name of the remote LU alias configured in Host Integration Server.

Mode name

When APPC Connection is selected, this field is the APPC mode and must be set to a value that matches the host configuration and SNA Server configuration.

Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security).

The default is typically QPCSUPP.

Distributed transactions

When this option is checked, two-phase commit (distributed unit of work) is enabled. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync service. This option works only with DB2 for OS/390 v5R1 or later.

DB2 Database Screen

Initial catalog

This field is the first entry in the Database section of the Connection properties.

This OLE DB property is used as the first part of a 3-part fully qualified table name.

In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual.

In DB2/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, one can be created using the Add option.

In DB2 Universal Database, this property is referred to as DATABASE.

If the provider supports changing the catalog for an initialized data source, the consumer can specify a different catalog name through the DBPROP_CURRENTCATALOG property in the DBPROPSET_DATASOURCE property set after initialization.

This is a required parameter.

This parameter is equivalent to the DBPROP_INIT_CATALOG OLE DB property ID.

Package collection

The name of the DRDA target collection (AS/400 library) where the Microsoft DAT should store and bind DB2 packages. This could be same as the Default Schema.

The DAT will create packages dynamically in the location to which the user points using the Package Collection parameter. By default, the DAT will automatically create one package in the target collection, if one does not exist, at the time the user issues their first SQL statement. The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the User ID value configured in the data source. The package is created for use by SQL statements issued under the same isolation level based on the Isolation Level value configured in the data source.

A problem can arise in multi-user environments. For example, if a user specifies a Package Collection value that represents a DB2 collection used by multiple users, but this user does not have authority to GRANT execute rights to the packages to other users (for example, PUBLIC), the package is created for use only by this user. This means that other users may be unable to access the required package. The solution is for an administrative user, with package administrative rights (for example, PACKADM authority in DB2 for OS/390), to create a set of packages for use by all users.

Default schema

The name of the Collection where the DAT looks for catalog information. The Default Schema is the "SCHEMA" name for the target collection of tables and views. The DAT uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection.

For DB2, the Default Schema is the target AUTHENTICATION (User ID or "owner").

For DB2/400, the Default Schema is the target COLLECTION name.

For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name.

If the user does not provide a value for Default Schema, the OLE DB Provider uses the USER_ID provided at logon. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. This default is inappropriate in many cases, therefore it is essential that the Default Schema value in the data source be defined.

This parameter is equivalent to the DBPROP_DB2OLEDB_DEFAULTSCH OLE DB property ID.

Default qualifier

The name of the schema (collection/owner) with which to fully qualify unqualified object names. This attribute allows the user to access database objects without fully-qualifying the objects using a collection (schema) qualifier. The DAT sends this value to DB2 using a SET CURRENT SQLID statement, instructing the DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statements. If you do not set a value for default qualifier, no SET statement is issued by the DAT. This OLE DB property is only valid when connecting to DB2 for MVS (OS/390, z/OS).

Mainframe File System (VSAM) Screen

Host column description file

The fully qualified file name of the Distributed Data Management (DDM) host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the system directory where the Host Integration Server server or client software was installed. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400.

This parameter is equivalent to the DBPROP_SNAOLEDB_HCDPATH OLE DB property ID.

AS/400 File System Screen

Location

The remote database name used for connecting to OS/400 systems. In DB2/400, this property is referred to as RDBNAM.

This parameter is not used when connecting to mainframe systems.

Default library

This parameter indicates the default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files.

This parameter is equivalent to the DBPROP_SNAOLEDB_LIBRARY OLE DB property ID.

Host file metadata assembly

The fully qualified file name of the Distributed Data Management (DDM) Host Column Description (HCD) file (created using the Data Access tool), or the fully qualified file name of a Host File Metadata Assembly generated by the Host File Designer in Visual Studio 2005.

This parameter can be a UNC string up to 256 characters in length.

This parameter is required when the data source information will be used with the BizTalk Adapter for Host Files or Managed Provider for Host Files, and is optional when connecting to an AS/400 using SNAOLEDB.

Adding a Column

This command displays the Column Properties Dialog box. You can also access this dialog box by selecting a column and clicking **Properties** from the context-sensitive menu.

To add a column

1. In the **Data Source Browser** window, click the **Action** menu.
2. Click **Add Column**. The **Column properties: General tab** appears.
3. Enter the **Name**, **Alias**, and **Comment** (optional). For more information on these properties, click the **Help** button. When you are finished, click **Apply**.
4. Click the **Host tab**. Select the appropriate **Type** and **CCSID**, and enter the **Length**, **Precision**, and **Scale**. For more information on these properties, click the **Help** button. When you are finished, click **Apply**.
5. Click the **Local tab**. Select the **Type** and check **Use qualifier** if appropriate. For more information on these properties, click the **Help** button. When you are finished, click **OK**.

Column properties: General

Name

The name of the column.

Alias

The alias for the column.

Comment

A comment (optional).

Column properties: Host

Type

The host data type, which determines how data is stored on the host.

CCSID

The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set to true, character data is converted based on the DB2 column CCSID and default ANSI code page.

This parameter defaults to U.S./Canada (37).

This parameter is equivalent to the DBPROP_DB2OLEDB_HOSTCCSID OLE DB property ID.

Length

The length of the data type on the host.**Precision**

The precision (or number of digits in a numeric type), of the data type on the host.

Scale

The scale (or number of decimal digits in a numeric type), of the data type on the host.

Column properties: Local

Type

The data type presented (such as DBTYPE_STR). Data conversions take place from the host type to the local type.

Use qualifier

Determines whether or not a qualifier is to be used.

Table properties

Name

Type a name for your table.

Use table for file transfer

Selecting this option allows you to enter additional fields for your table, below.

Field delimiter

Enter the field delimiter for your table.

Record delimiter

Enter the record delimiter for your table.

Text qualifier

Enter the text qualifier for you table.

File creation type

Select Direct, Sequential, or Indexed from the list.

Sort

If you selected a File Creation Type of Indexed, you have the option of sorting in either Ascending or Descending order.

Key position

If you selected a File Creation Type of Indexed, you have the option of specifying a key position.

Key length

If you selected a File Creation Type of Indexed, you have the option of specifying a key length.

Options Property Page

The Data Access Tool configures the following default values at installation.

Files	Location
OLE DB UDL files	C:\My Documents\Host Integration Projects\Data Sources\
ODBC File DSNs	C:\Program Files\Common Files\ODBC\Data Sources
Host column description files	C:\My Documents\Host Integration Projects\Data Descriptions

Click any of the **Browse** buttons to select a new directory.

DB2 Locale Screen

Host CCSID

The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set to true, character data is converted based on the DB2 column CCSID and default ANSI code page.

This parameter defaults to U.S./Canada (37).

This parameter is equivalent to the DBPROP_DB2OLEDB_HOSTCCSID OLE DB property ID.

PC code page

This parameter indicates the code page to be used on the personal computer for character code conversion. This parameter is required when processing binary data as character data. Unless the **Process Binary as Character** checkbox is selected (value is set to true), character data is converted based on the default ANSI code page configured in Windows.

This parameter defaults to Latin 1 (1252).

This parameter is equivalent to the DBPROP_DB2OLEDB_PCCODEPAGE OLE DB property ID.

Process binary as character

When this option is checked, it indicates that binary data fields should be processed as characters. This option treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters.

This parameter is equivalent to the DBPROP_DB2OLEDB_BINASCHAR OLE DB property ID.

Mainframe and AS/400 Locale Screen

Host CCSID

The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set to true, character data is converted based on the DB2 column CCSID and default ANSI code page.

This parameter defaults to U.S./Canada (37).

This parameter is equivalent to the DBPROP_DB2OLEDB_HOSTCCSID OLE DB property ID.

PC code page

This parameter indicates the code page to be used on the personal computer for character code conversion. This parameter is required when processing binary data as character data. Unless the **Process Binary as Character** checkbox is selected (value is set to true), character data is converted based on the default ANSI code page configured in Windows.

This parameter defaults to Latin 1 (1252).

This parameter is equivalent to the DBPROP_DB2OLEDB_PCCODEPAGE OLE DB property ID.

Process binary as character

When this option is checked, it indicates that binary data fields should be processed as characters. This option treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters.

This parameter is equivalent to the DBPROP_DB2OLEDB_BINASCHAR OLE DB property ID.

Security Screen

Interactive sign-on

Authentication to a data source can be done using interactive sign-on or Single Sign-On. For interactive sign-on, the user name and password must match host credentials.

User name

A valid user name and password are normally required to access data sources on a host. These values are case sensitive. Do not select the **Single Sign-On** option if a specific user name and password are to be entered.

Password

A valid user name and password are normally required to access data sources on hosts. These values are case sensitive.

Allow saving password

You have the option of saving the password in the .udl file by selecting this check box. Users and administrators should be warned that this option saves the authentication information (password) in plain text within the .udl file. For security purposes, however, it is strongly recommended that you set these properties manually, instead of using a .udl. Note that if you select this option when using the ODBC Driver for DB2, and then uninstall and reinstall Host Integration Server, the password will be erased.

Single Sign-On

Select this check box to enable using the Host Integration Security features providing a Single Sign-On to access this data source.

When this check box is selected, the **User name** and **Password** fields are dimmed and become inaccessible. The user name and password fields are set based on the Windows 2000 logon.

When this check box is not selected, the **User name** and **Password** fields must normally contain appropriate values to access data sources on hosts.

Affiliate Application

If you selected Single Sign-On, choose an Affiliate Application from the list. The Enterprise Single Sign-On (SSO) Affiliate applications are logical entities that represent a system or sub-system such as a host, back-end system, or line of business application to which you are connecting using SSO. An affiliate application can represent a back-end system such as a mainframe or UNIX computer. It can also represent an application such as SAP, or a subdivision of the system, such as the "Benefits" or "Pay stub" sub-systems.

DB2 Validation Screen

Use this screen to validate your configuration.

Click **Connect** to perform a test connection.

Click **Packages** to create the DB2 packages required for executing SQL statements.

Click **Sample Query** to retrieve a list of tables in the default database collection.

Mainframe and AS/400 Validation Screen

Use this screen to validate your configuration.

Click **Connect** to test the connection to the mainframe or AS/400 system.

Click **Sample Query** to retrieve a list of files in the default library.

DB2 Saving Information Screen

Use this screen to name and save your configuration.

Data source name
Enter a name of your choice. This name is used when saving the data source configuration to one or more data source types.

OLE DB or Managed group, **ODBC** group
Make the appropriate selection(s) for your data source.

Mainframe and AS/400 Saving Information Screen

Use this screen to name and save your configuration.

Data source name
Enter a name of your choice. This name is used when saving the data source configuration to one or more data source types.

Universal data linkand **Initialization string file**
Make the appropriate selection(s) for your data source.

Advanced Options Screen

The settings on this screen are optional.

Connection pooling
Selecting this option will cache connections for reuse.

Cache authentication
This parameter determines whether the OLE DB Provider for AS/400 and VSAM caches authentication information, such as a password, in an internal cache. This parameter is not currently supported by the OLE DB Provider for AS/400 and VSAM and defaults to false.

This parameter is equivalent to the DBPROP_CACHE_AUTHINFO OLE DB property ID.

Mode (Read/ReadWrite)
Determines whether the connection is read-only, or if it allows modifications to the database.

Repair host keys
This parameter provides for repair of invalid key offsets received from OS/400 when keys have been defined using the DDS "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry.

This parameter defaults to false. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.

Read only
Selecting this option prevents a data source from being updated.

Alternate TP Name
This is the transaction program name on a DB2 DRDA application server.

Strict Validation
Select this option if desired.

Finish Screen

This screen displays a summary and status of your configuration. Click **Finish** to implement your actions.

Data User Interface Elements

The topics in this section describe the elements in the Data Integration User Interface.

In This Section

Data Source Browser

Data Source Folder

Data Source Item

Data Descriptions Folder

Data Description File

Data Description Table

Data Description Column

Table Properties

Data Source Browser

The Data Source Browser is the window where you configure and manage your data sources.

The window is divided into three parts: a tree view containing the data sources and data descriptions, a list view containing details of a selected node, and a result view containing the text results of an action (such as a package creation).

The procedures in the following topics describe how to manage data. Many commands are also accessible through a context-sensitive menu which appears when you right-click any section of the Browser window. In addition, the F5 key refreshes the tree view, the Delete key deletes the currently selected item, and the F1 key opens the online Help.

Data Source Folder

This folder contains data sources and groups of data sources.

Data Source Item

Right-clicking a data source allows you to view, edit, test, delete, or rename it.

Data Descriptions Folder

This folder displays the names of all configured data descriptions. Click the individual files to view their properties.

Data Description File

Right-click a data description to view its properties, or to edit or delete the file.

Data Description Table

A Data description table contains the meta-data describing a file or table on a mainframe, AS/400, or System 36 file system. A table description holds information about the various columns in that table.

Data description tables are stored in "Host Column Description" files and can be found under the "Host Column Descriptions" folder in the DAT browser.

Data Description Column

A Data description column contains the meta-data describing a field or column of a mainframe, AS/400, or System 36 file system table or file. This meta-data includes the column name, its host type, and its local type.

Data description columns are stored in "Host Column Description" files and can be found under the "Host Column Descriptions" folder in the DAT browser.

Table Properties

Displays the properties for the selected table.

Configuring a Data Source

The following sections contain information about configuring a data source.

In This Section

Configuring a Data Source for OLE DB Provider for AS/400 and VSAM

Configuring a Data Source for the ODBC Driver for DB2

Configuring a Data Source for OLE DB Provider for AS/400 and VSAM

You must configure data source information for each AS/400 or mainframe system data source object that is to be accessed using OLE DB Provider for AS/400 and VSAM. The default parameters for OLE DB Provider for AS/400 and VSAM are used only when these parameters are not configured for each data source.

Microsoft Data Link, a core element of Microsoft Data Access Components (MDAC), provides a uniform method for creating file-persistent OLE DB data source object definitions in the form of universal data link (.udl) files. The Data Source Wizard in the Microsoft Data Access Tool can help define UDL files. OLE DB consumer applications, such as Data Transformation Services in Microsoft SQL Server can use the UDL files to connect to IBM data sources, such as DB2 and the mainframe file system.

To configure a data source for OLE DB Provider for AS/400 or VSAM

You must create a data link to configure parameters for your OLE DB data source. You can create a new data link by clicking the shortcut in the Host Integration Server program folder.

The properties of a universal data link file can be edited by opening the file from Windows Explorer.

The **Provider** tab lets you select the OLE DB provider (the provider name string) to be used in this .udl file from a list of possible OLE DB providers. Select **Microsoft OLE DB Provider for AS/400 and VSAM**.

The **Connection** tab lets you configure the basic properties required to connect to a data source. For **OLE DB Provider for AS/400 and VSAM**, the connection properties include the following values:

Property	Description
Data Source	This is an optional parameter that can be used to describe the data source.
Network	This drop-down list allows for selecting the type of network connection to use. The options are **TCP/IP Connection** or **SNA Connection**. If **TCP/IP Connection** is selected, click **More Options**, to open a dialog box for configuring TCP/IP network settings. The parameters you can configure include the IP address of the remote host (or a host name alias for this computer) and the Network Port (TCP/IP port) used to communicate with the host. The default value for the Network Port is 446. The IP address of the host has no default value. If **SNA Connection** is selected (using LU 6.2), click **More Options** to open a dialog box for configuring SNA network settings. The parameters you can configure include the following: the APPC local LU alias, the APPC remote LU alias, and the APPC mode used to communicate with the host. The default value for the APPC mode ordinarily uses QPCSUPP. The local and remote LU alias fields do not have default values.
Single Sign-On	This option enables using the Host Integration Security features providing a single sign-on to access this OLE DB data source. When this option is selected, the **User name** and **Password** fields are dimmed and become inaccessible. The **User name** and **Password** fields are set based on the Windows logon values. When this option is not selected, the **User name** and **Password** fields ordinarily contain appropriate values to access data sources on hosts.
User name	A valid user name and password are ordinarily required to access data sources on hosts. These values are case sensitive. The user must click the option button that requires a specific user name and password to be entered.
Password	A valid user name and password are ordinarily required to access data sources on hosts. These values are case sensitive. The user will have to clear the **Blank password** check box, if it is selected, to enter a password. Optionally, the user can choose to save the password in the .udl file by selecting the **Allow saving password** check box. Users and administrators should be warned that this option persists the authentication information in plain text within the .udl file.

Location	The remote database name used for connecting to OS/400 systems. In DB2/400, this property is referred to as RDBNAM. This parameter is not used when connecting to mainframe systems.
Default Library	This parameter indicates the default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files. This parameter is equivalent to the DBPROP_SNAOLEDB_LIBRARY OLE DB property ID.
Host Column Description File	The fully qualified file name of the Distributed Data Management (DDM) host column description (HCD) file. This parameter can be a UNC string up to 256 characters long. A path does not must be included in the name if the HCD file is located in the system directory where the Host Integration Server server or client software was installed. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400. This parameter is equivalent to the DBPROP_SNAOLEDB_HCDPATH OLE DB property ID.

The **Connection** tab also includes a **Test Connection** button that can be used to test the connection parameters. The connection can only be tested after all the required parameters are entered. When this button is selected, a session will be established to the host computer using OLE DB Provider for AS/400 and VSAM.

The **Advanced** tab exposes OLE DB standard properties. For **OLE DB Provider for AS/400 and VSAM**, the advanced properties include the following values:

Property	Description
Host CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set to true, character data is converted based on the DB2 column CCSID and default ANSI code page. By default, this parameter uses U.S./Canada (37). This parameter is equivalent to the SNAOLEDB_HOSTCCSID OLE DB property ID.
PC Code Page	The *PC Code Page* parameter indicates the code page to be used on the personal computer for character code conversion. This parameter is required when processing binary data as character data. Unless the **Process Binary as Character** check box is selected (value is set to true), character data is converted based on the default ANSI code page configured in Windows. By default, this parameter uses Latin 1 (1252). This parameter is equivalent to the DBPROP_DB2OLEDB_PCCODEPAGE OLE DB property ID.
Read only	When the *Read Only* parameter is checked in the **Advanced** tab, the OLE DB Provider for AS/400 and VSAM creates a read-only data source by setting the *Mode* parameter to Read (DB_MODE_READ). A user has read access to files and cannot do update operations.
Repair Host Keys	This parameter provides for repair of invalid key offsets received from OS/400 when keys have been defined using the DDS "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry. By default, this parameter is false. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.
Process Binary as Character	When this option is checked (property is set to true), the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. By default, this parameter is false. This parameter is equivalent to the SNAOLEDB_BINASCHAR OLE DB property ID.

The **All** tab lets you configure additional properties used to connect to a data source. Some of the properties in the **All** tab are required. These properties can be edited by selecting a property from the displayed list and selecting **Edit Value**. For Microsoft OLE DB Provider for AS/400 and VSAM, these properties include the following values:

Property	Description
APPC Local LU Alias	The name of the local LU alias configured in the Host Integration Server computer. This parameter is equivalent to the DBPROP_SNAOLEDB_LOCALLU OLE DB property ID.
APPC Mode Name	When LU 6.2 (SNA) is selected for the Network Transport Library, this field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server computer configuration. Valid values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also valid: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). By default, this parameter ordinarily is QPCSUPP. This parameter is equivalent to the DBPROP_SNAOLEDB_APPCMODE OLE DB property ID.
APPC Remote LU Alias	When LU 6.2 (SNA) is selected for the Network Transport Library, this field is the name of the remote LU alias configured in the Host Integration Server computer. This parameter is equivalent to the DBPROP_SNAOLEDB_LOCALLU OLE DB property ID.
Cache Authentication	This parameter determines whether the OLE DB Provider for AS/400 and VSAM caches authentication information, such as a password, in an internal cache. This parameter is not currently supported by the OLE DB Provider for AS/400 and VSAM and is false by default. This parameter is equivalent to the DBPROP_CACHE_AUTHINFO OLE DB property ID.
Connection Timeout	The time (in seconds) to wait for initialization to finish. This parameter is not currently supported by the OLE DB Provider for AS/400 and VSAM and is 0 by default. This parameter is equivalent to the DBPROP_INIT_TIMEOUT OLE DB property ID.
Data Source	This is an optional parameter that can be used to describe the data source. This parameter is equivalent to the DBPROP_INIT_DATASOURCE OLE DB property ID.
Default Library	This parameter indicates the default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files. This parameter is equivalent to the DBPROP_SNAOLEDB_LIBRARY OLE DB property ID.
Encrypt Password	This parameter determines whether special security mechanisms are used to ensure password privacy. This parameter is not currently supported by the OLE DB Provider for AS/400 and VSAM and is false by default. This parameter is equivalent to the DBPROP_AUTH_ENCRYPT_PASSWORD OLE DB property ID.

Extended Properties	This parameter is a string that contains provider-specific, extended connection information. Properties passed through this parameter should be delimited by semicolons and are interpreted by the OLE DB provider's underlying network client. The use of this property implies that the OLE DB consumer knows how this string will be interpreted and used by the OLE DB provider. This parameter should be used only for provider-specific connection information that cannot be explicitly described through the other property parameters. This parameter is equivalent to the DBPROP_INIT_PROVIDERSTRING OLE DB property ID.
Host CCSID	The character code set identifier (CCSID) matching the data as represented on the host. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the host column CCSID and default ANSI code page. By default, this parameter is U.S./Canada (37). This parameter is equivalent to the DBPROP_SNAOLEDB_HOSTCCSID OLE DB property ID.
Host Column Description File	The fully qualified file name of the Distributed Data Management (DDM) host column description (HCD) file. This parameter can be an UNC string up to 256 characters long. A path does not have to be included in the name if the HCD file is located in the system directory where the Host Integration Server Server or Client software was installed. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400. This parameter is equivalent to the DBPROP_SNAOLEDB_HCDPATH OLE DB property ID.
Impersonation Level	This parameter indicates the level of impersonation that the server can use when impersonating the client computer. This property applies only to network connections other than remote procedure call (RPC) connections; these impersonation levels are similar to those provided by RPC. The values of this property correspond directly to the levels of impersonation that can be specified for authenticated RPC connections, but can be applied to connections other than authenticated RPC. This parameter can be set to one of the following values: Anonymous — The client computer is anonymous to the server. The server process cannot obtain identification information about the client computer and cannot impersonate the client computer. Delegate — The process can impersonate the client's security context while acting on behalf of the client. The server process can also make outgoing calls to other servers while acting on behalf of the client. Identity — The server can obtain the client's identity. The server can impersonate the client computer for access control list (ACL) checking, but cannot access system objects as the client computer. Impersonate — The server process can impersonate the client's security context while acting on behalf of the client computer. This information is obtained when the connection is established, not on every call. By default, this parameter uses Impersonate. This parameter is equivalent to the DBPROP_INIT_IMPERSONATION_LEVEL OLE DB property ID.
Integrated Security	This parameter is a string that contains the name of the authentication service used by the server to identify the user using the identity provided by an authentication domain. For example, for Microsoft® Windows 2000 Integrated Security, this is Security Support Provider Interface (SSPI). If this parameter is a null pointer, the default authentication service should be used. When this property is used, no other DBPROP_AUTH* properties are needed and, if provided, their values are ignored. This parameter is equivalent to the DBPROP_AUTH_INTEGRATED OLE DB property ID.
Locale Identifier	This parameter specifies the locale to use. This parameter is not supported by the OLE DB Provider for AS/400 and VSAM and is 437 by default. This parameter is equivalent to the DBPROP_INIT_LCID OLE DB property ID.

Location	This parameter indicates the remote database name used for connecting to OS/400 systems. In DB2/400, this property is referred to as RDBNAM. This parameter is not used when connecting to mainframe systems. This parameter is equivalent to the DBPROP_INIT_LOCATION OLE DB property ID.
Mask Password	This parameter indicates whether the password should be sent to the data source or enumerator in a masked form. This parameter is not supported by the OLE DB Provider for AS/400 and VSAM and defaults to false. This parameter is equivalent to the DBPROP_AUTH_MASK_PASSWORD OLE DB property ID.
Mode	After a connection is established, this parameter represents a bit mask of the access permissions that will be applied to the data file. As implemented by the OLE DB Provider for AS/400 and VSAM, access permissions apply to host file locks and do not apply to record locks. The allowable values include the following: Read, ReadWrite, Share Deny None, Share Deny Read, Share Deny Write, Share Exclusive, and Write. This parameter can be a combination of zero or more of the following: DB_MODE_READ — Read-only. DB_MODE_WRITE — Write-only. DB_MODE_READWRITE — Read/write (DB_MODE_READ | DB_MODE_WRITE). DB_MODE_SHARE_DENY_READ — Prevents others from opening in read mode. DB_MODE_SHARE_DENY_WRITE — Prevents others from opening in write mode. DB_MODE_SHARE_EXCLUSIVE — Prevents others from opening in read/write mode (DB_MODE_SHARE_DENY_READ | DB_MODE_SHARE_DENY_WRITE). DB_MODE_SHARE_DENY_NONE — Neither read nor write access can be denied to others. This parameter is equivalent to the DBPROP_INIT_MODE OLE DB property ID.
Network Address	When TCP/IP has been selected for the Network Transport Library, this parameter is used to locate the target host computer. This parameter indicates the IP address or TCP/IP host name alias associated with the DDM server on the host. The network address is required when connecting through TCP/IP. This parameter is equivalent to the DBPROP_SNAOLEDB_NETADDRESS OLE DB property ID.
Network Port	When TCP/IP has been selected for the Network Transport Library, this parameter is used to locate the target DDM service access port when connecting through TCP/IP. This parameter represents the TCP/IP port used for communication with the DDM service on the host. The default value is TCP/IP port 446. This parameter is equivalent to the DBPROP_SNAOLEDB_NETPORT OLE DB property ID.
Network Transport Library	This parameter, which represents the dynamic-link library used for transport, designates whether the OLE DB provider connects through SNA LU 6.2 or TCP/IP for network communication. The possible values for this parameter are TCPIP or SNA. If TCPIP is selected, then values for Network Address and Network Port are required. TCP/IP connectivity to the mainframe is not supported by the OLE DB Provider for AS/400 and VSAM. If SNA is selected, then values for APPC Local LU Alias, APPC Mode Name, and APPC Remote LU Alias are required. This value defaults to SNA. This parameter is equivalent to the DBPROP_SNAOLEDB_NETTYPE OLE DB property ID.
Password	A valid user name and password are normally required to access data sources on hosts. The password is case-sensitive and is shown as asterisks in this dialog box for security purposes. Optionally, you can choose to save the password in the .udl file by clicking the **Allow saving password** check box. Users and administrators should be warned that this option persists the authentication information in plain text within the .udl file. This parameter is equivalent to the DBPROP_AUTH_PASSWORD OLE DB property ID.

PC Code Page	This parameter indicates the code page used for character code conversion. This property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the default ANSI code page configured in the Windows operating system. If this parameter is set to **Binary** or **65535**, then no character code conversions take place. This parameter defaults to Latin 1 (1252). This parameter is equivalent to the DBPROP_SNAOLEDB_PCCODEPAGE OLE DB property ID.
Persist Security Info	This parameter indicates whether the data source object is allowed to persist sensitive authentication information such as a password along with other authentication information. Optionally, a user can choose to save the password in the .udl file by clicking the **Allow saving password** check box. Users and administrators should be warned that this option persists the authentication information in plain text within the .udl file. This parameter defaults to false. This parameter is equivalent to the DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO OLE DB property ID.
Process Binary as Character	This parameter indicates whether to **Process Binary as Character** fields (CCSID of 65535) as character data type fields on a per data source basis. The **Host CCSID** and **PC Code Page** values are required input parameters when this parameter is true. The default for this parameter is false. Do not process binary fields as character fields. This parameter is equivalent to the DBPROP_SNAOLEDB_BINASCHAR OLE DB property ID.
Protection Level	This parameter indicates the level of protection of data sent between client computer and server. The values of this property correspond directly to the levels of protection that can be specified for authenticated RPC connections. This parameter can be set to one of the following values: DB_PROT_LEVEL_NONE — Performs no authentication of data sent to the server. DB_PROT_LEVEL_CONNECT — Authenticates only when the client computer establishes the connection with the server. DB_PROT_LEVEL_CALL — Authenticates the source of the data at the beginning of each request from the client computer to the server. DB_PROT_LEVEL_PKT — Authenticates that all data received is from the client computer. DB_PROT_LEVEL_PKT_INTEGRITY — Authenticates all data received is from the client computer and that it has not been changed in transit. DB_PROT_LEVEL_PKT_PRIVACY — Authenticates all data received is from the client computer, that it has not been changed in transit, and protects the privacy of the data by encrypting it. This parameter is not supported by the OLE DB Provider for AS/400 and VSAM and defaults to the connect level of protection. This parameter is equivalent to the DBPROP_INIT_PROTECTION_LEVEL OLE DB property ID.
Read only	When the *Read Only* parameter is checked in the Advanced tab, the OLE DB Provider for AS/400 and VSAM creates a read-only data source by setting the *Mode* parameter to Read (DB_MODE_READ). A user has read access to files and cannot do update operations.
Repair Host Keys	This parameter provides for repair of invalid key offsets received from OS/400 when keys have been defined using the DDS "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry. This parameter defaults to false. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.

Strict Validation	This parameter indicates whether strict validation should be used and defaults to false. This parameter is equivalent to the DBPROP_SNAOLEDB_STRICTVAL OLE DB property ID.
User ID	A valid user name is normally required to access data sources on hosts. This value is case-sensitive. This parameter is equivalent to the DBPROP_AUTH_USERID OLE DB property ID.

Configuring a Data Source for the ODBC Driver for DB2

To configure a data source for the ODBC Driver for DB2

1. In Windows, click **Start**, click **Settings**, and then click **Control Panel**.
2. Double-click the **ODBC** icon to display the **ODBC Data Source Administrator** dialog box.
3. If you are configuring an existing data source, select the data source name and click **Configure** to display the **Microsoft ODBC Driver for DB2 Configuration** dialog box.
4. If you are configuring a new data source click either the **User DSN** tab, the **File DSN** tab, or the **System DSN** tab. Click **Add**, select **Microsoft ODBC Driver for DB2 Driver**, and then click **Finish** to display the **Microsoft ODBC Driver for DB2 Configuration** dialog box.
5. Type the appropriate values in the fields, and then click **Apply**.

The **Microsoft ODBC Driver for DB2 Configuration** dialog box contains the following five tabs:

The **General** tab contains the following fields:

Parameter	**Comments**
Data Source Name	A blank field for specifying the name of the data source. Enter a string that identifies this ODBC data source. The data source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file, which is stored in C:\Program Files\Common Files\ODBC\Data Sources.
Description	A blank field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.

The **Connection** tab allows the user to configure the basic attributes required to connect to a data source. For the Microsoft ODBC Driver for DB2, the **Connection** tab has the following fields:

Parameter	**Comments**
APPC Connection and TCP/IP Connection	An option button (radio button) is used to select the network transport. Valid options are APPC Connection (SNA LU 6.2) or TCP/IP Connection. For the default, APPC Connection, the values for APPC local LU alias, APPC remote LU alias, and APPC Mode Name are required. For TCP/IP Connection, the values for IP address and Network port are required.
APPC local LU alias	When APPC Connection is selected, this field is the name of the local LU alias configured in Host Integration Server.
APPC remote LU alias	When APPC Connection is selected, this field is the name of the remote LU alias configured in Host Integration Server.

APPC mode name	When APPC Connection is selected, this field is the APPC mode and must be set to a value that matches the host configuration and SNA Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). The default is typically QPCSUPP.
IP address	When TCP/IP Connection is selected as the network transport, this field indicates the IP address of the host DB2 server.
Network port	When TCP/IP Connection is selected as the network port, this field indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default is **IP port 446**.

The **Connection** tab also includes a **Test connection** button that may be used to test the connection parameters. A connection can only be tested after all of the required parameters for the **Connection** tab and other ODBC data source parameters are configured properly. When this button is clicked, a session is established with the remote DB2 system using ODBC Driver for DB2.

The **Security** tab allows the user to configure optional attributes used to restrict connections to a data source. For the Microsoft ODBC Driver for DB2, the **Security** tab has the following fields:

Parameter	Comments
Authentication	An option button (radio button) is used to select the type of authentication. Valid options are **Use this username** or **Use Single Sign-On**. For the default **Use this username** option, the value for the user name is required.
Use this username	When this option is selected, authentication is based on the user name entered in the textbox. A valid user name is normally required to access data on DB2. A user name can remain optionally in the DSN. The ODBC Driver for DB2 will prompt the user at run-time to enter a valid password. Additionally, the prompt dialog box will enable the user to override the user name that is stored in the DSN.
Use Single Sign-On	An option button to select whether Single Sign-On or a specific user name should be used. Single Sign-On is an optional Host Security feature. Single Sign-On enables the administrator to create data source definitions that isolate the logon process from the end user. The user context for Single Sign-On is the user context associated with the SNA DB2 service.

The AS/400 computer is case-sensitive with regard to user IDs and passwords. When connecting to DB2 for OS/400, user names and passwords must be in uppercase. The AS/400 only accepts a DB2 for OS/400 user ID and password in uppercase. If a DB2 for OS/400 connection fails due to incorrect authentication, the ODBC driver resends the authentication, forcing the user ID and password into uppercase.

When connecting to DB2 on IBM mainframes, user names and passwords can be of mixed case; the mainframe is not case-sensitive. The ODBC driver sends these values in uppercase.

DB2 Universal Database (UDB) is case sensitive. The user ID is stored in uppercase. The password is stored in mixed case and users must enter the password in the correct case. The ODBC driver sends the password exactly in the case entered by the user. The user ID should contain only the user name, not a combination of the Windows domain name and username.

The **Target Database** tab allows the user to configure required, as well as optional, attributes used to define the target DB2 system. For the ODBC Driver for DB2, the **Target Database** tab has the following fields:

Parameter	Comments
Initial catalog	This parameter is used as the first part of a three-part fully qualified DB2 table name. It is referred to by different names depending on the DB2 platform. In DB2 for OS/390 and DB2 for MVS, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 that you need to connect to on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then a value can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE.
Package collection	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This can be the same as the default schema. The ODBC Driver for DB2, which is implemented as an IBM DRDA Application Requester, uses packages to issue dynamic and static SQL statements. The ODBC driver creates packages dynamically in the location that the user points to using the Package Collection parameter. By default, the ODBC Driver for DB2 automatically creates one package in the target collection, if one does not exist, at the time the user issues their first SQL statement. The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the user ID value configured in the data source. The package is created for use by SQL statements issued under the same isolation level based on the Isolation Level value configured in the data source. Problems can arise in multi-user environments. For example, if a user specifies a Package Collection value that represents a DB2 collection used by multiple users, but this user does not have authority to GRANT execute rights to the packages to other users (for example, PUBLIC), then the package is created only for use by this user. This means that other users may be unable to access the required package. The solution is for an administrative user, with package administrative rights (for example., PACKADM authority in DB2 for OS/390), to create a set of packages for use by all users. The ODBC Driver for DB2 ships with two utility programs for use by administrators to create packages. The crtpkg.exe tool is a command line utility for the administrator to create packages. The crtpkgw.exe tool is a Windows GUI utility used for the same purpose. Either of these utilities can be run using a privileged user ID to create packages in collections accessed by multiple users. These utilities will create sets of packages and grant EXECUTE privilege to PUBLIC for all (see descriptions under the Default Isolation parameter). The packages created are as follows: AUTOCOMMIT package (MSNC001 is only applicable on DB2/400) READ_UNCOMMITTED package (MSUR001) REPEATABLE_READ package, (MSRS001) READ_COMMITTED package, (MSCS001) SERIALIZABLE package (MSRR001). After being created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, the DB2 for OS/400 QSYS2.SYSPACKAGE, and the DB2 Universal Database (UDB) SYSIBM.SYSPACKAGE catalog tables.
Default schema	The name of the Collection where the ODBC Driver for DB2 looks for catalog information. The Default schema is the SCHEMA name for the target collection of tables and views. The ODBC driver uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (for example, ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or owner). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, the ODBC driver uses the USER_ID provided at logon. For DB2/400, the driver uses QSYS2 if no collection is found matching the USER_ID value. This default is inappropriate in many cases so it is essential that the Default Schema value in the data source be defined.
DBMS Platform	The target DB2 platform property value is used to optimize performance of the ODBC driver when executing operations such as data conversion. The default value is DB2/MVS.

Default Qualifier	The name of the schema (collection/owner) with which to fully qualify unqualified object names. This attribute allows the user to access database objects without fully-qualifying the objects using a collection (schema) qualifier. The ODBC driver sends this value to DB2 using a SET CURRENT SQLID statement, instructing the DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statements. If you do not set a value for default qualifier, then no SET statement is issued by the ODBC driver. This ODBC connection attribute is only valid when connecting to DB2 for MVS (OS/390, z/OS).
Alternate TP Name	The remote transaction program name is optional. For example, it is used when configuring an off-line DB2 demo link service connection.
Distributed transactions	When this option is checked, two-phase commit (distributed unit of work) is enabled. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and one of the HIS Resync services.
Process binary as character	When this option is checked, it indicates that binary data fields should be processed as characters. This option treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. See the **Locale** tab.

The **Locale** tab allows the user to configure the parameters used for character conversion between the client computer and the DB2 server. Two versions of the **Locale** tab are possible depending on which versions of SNA Server client software and ODBC for DB2 Driver client software are installed.

For the ODBC Driver for DB2, the **Locale** tab has the following fields:

Parameter	**Comments**
Host CCSID	The coded character set identifier (CCSID) matching the DB2 data as represented on the remote computer. This property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the DB2 column CCSID and configured ANSI code page. This parameter defaults to U.S./Canada (37).
PC code page	This parameter indicates the personal computer code page to use. It is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the default ANSI code page configured in Windows. The default value for this property is Latin 1 (1252).

Click **OK** or **Cancel** when data entry is finished. If you click **OK**, the values specified become the defaults when an application connects to this data source. These default values can be changed at any time using this procedure to reconfigure the data source. An ODBC application can override these defaults by connecting to the data source using a connection string with alternate values.

Network Integration Help

Use the topics in this section to navigate through the Network Integration user interface.

In This Section

Print Service Properties

Click **No Event Log for Skipping Transparent Section** to prevent an entry in the Event Log every time print services skips a transparent section found while printing a host print job. Skipping transparent sections can be enabled from the **Advanced** tab of a print session properties page.

3270

Click **Flush Final FF** causes Host Print services to explicitly form feed the document at the end of a print job. This is not needed for most print jobs and configurations.

Click **Use Proportional Font Change** to prevent overlapping characters in documents containing nonfixed-type fonts printed through Host Print services. Most standard 3270 data streams are designated for fixed-type fonts.

Click **Ignore Characters 3F and Under** to cause the print service to ignore Hexadecimal characters 3F and below. In LU 1 data streams, this option causes hexadecimal characters below 40 to be replaced by spaces.

Click **Delay Print Start** to delay the start of the print job until printable data is received by Host Print services. In a few environments, hosts may poll the Host Print service for activity by sending "empty jobs". In such cases, not selecting this option may cause blank pages to be printed between actual print jobs.

LU 3 Only

Click **Do All FF** to force the printer driver to honor all form feed commands. (In most jobs, this is unnecessary and may lead to unwanted blank pages.)

Click **Always do NL** to insert a new line when the print services determines that the Maximum Print Position has been reached for a particular line of data. Selecting this option will prevent data from printing over an existing line.

Click **No Space After FF** to prevent print services from inserting a space character following a form feed.

Click **Support Embedded Printer Codes** to allow unmarked embedded characters, such as EBCDIC X'27' characters, to pass through print services to the printer.

LU 1 Only

Click **Use Fixed Tabs** to disable normal tab functionality (such as aligning tabs in columns) and interpret each tab as a fixed number of spaces. The number of spaces for a tab stop is based on the size of the first tab in the Set Horizontal Format SCS control code.

APPC

Set the **Activation Retry Limit** to the number of times print services will attempt to activate the APPC conversation following a terminated connection. The default value is –1 (infinite), meaning that attempts to activate the session will continue until it is successful or terminated manually.

Click **Infinite Retry Limit** to force the system to retry printing the job until it is successful.

Set the **Activation Retry Interval** to the number of seconds to wait before trying to print the job again. The default value is 10 seconds.

Print Session Properties

The following tabs are available on the Print Session 3270 Properties sheet:

Print Session Properties: General

Session Name

To print 3270 jobs, you must define the session name, which is a descriptive name to distinguish different printers on the network.

Comment

Optionally, type a comment of 25 characters or less.

Session Activation

Click **Manual** to activate the print session manually, or **Automatic** to activate the print session automatically when the Host Print service is started.

Host Code Page

Click **Language**, and then select a **languagecode** from the drop-down list.

If your language code does not appear in the drop-down list, click **Custom**, then click on the **File** button. The **Select Custom Page File** dialog box appears. Select your custom code page and click **Open**.

Note
The custom page support in Host Print service provides Arabic and Hebrew code page support for left-to-right output only. T he Host Print service does not support bi-directional data streams from right to left.

Destination

The default printer should appear in the box. Otherwise, click **Printer** to open the **Print Setup** dialog box. Make your selections and click **OK**.

Click **File** to send your job to a file. Check **Unique** to activate the **Reset** button. This feature allows you to reset the print job sequential numbering scheme (for example, when the number gets too high). Click **Reset**, and then click **Yes** to reset to the printer file extension numbering.

Note
You need to select a printer driver to print to a file. Select a valid printer driver, then select File in the Destination box. Otherw ise, you get an error message.

Print Session Properties: 3270

LU Name

Choose a **3270 printer LU** for the print session from the **LU Name** drop-down list. The drop-down list shows all of the LUs in an SNA Subdomain. If no 3270 printer LUs appear, you need to define one.

Job Termination When

Set job termination to either When End Bracket Received or When Unbind Received.

Selecting **When End Bracket Received** means the Host Print service will treat receiving the End Bracket notification from the host as an indication that the job is complete. Otherwise, the Print service defaults to spooling the job until the session ends: **When Unbind Received**.

Note
If the host application sends print jobs that comprise multiple SNA brackets, set job termination to When End Bracket Receiv ed.

Job Timeout

To set a parameter for terminating print jobs, click the **Timeout Job After** box. Click the **Seconds Inactivity** up and down buttons to set the time limit for terminating print jobs. In some cases, an LU 3 print job is sent down from the host over an

extended period of time. Selecting a Job Timeout ensures that the portions of the job are regularly form-fed. This option also provides normal timeout functionality for other 3270 print jobs.

Monitor Job

Click **Request Definite Response** if you want to invoke an advanced feature for applications that require a high level of assurance that a job completed. This feature sends a message to the host stating that the print job completed.

Note
If you select the Request Definite Response feature, the host job must mark the data as RQD (definite response required).

Print Session Properties: Job Format

Do Not Format Print Job

Do Not Format Print Job allows print jobs that are formatted with host software to bypass the Microsoft® Windows®-based printing format system. The Windows-based Host Print service treats all received data as transparent. All data is passed directly to the printer.

Format Print Job

Print jobs from a host system can be formatted one of two ways:

GDI

The Windows Graphical Device Interface (GDI) is used to format the print job. SCS codes in the data are interpreted and represented, and the GDI automatically supports all the configurable options such as font, margins, duplex, paper-source etc. Transparent sections (for example, containing PCL escape sequences) are just ignored. When such data is discarded, a log is written to the Event Log, containing the first 32 bytes of discarded data. This logging can be globally disabled from the Host Print service properties page.

Selecting this option disables the Ignore **Transparent Sections for PDT Formatting** on the **Advanced** tab.

Note
If your print job has a mixture of SCS codes and PCL escapes, then you should use a PDT. The PDT should contain mappings f or the SCS code functionality, and the PCL escapes should pass directly to the printer without being mapped.

PDT

A Printer Definition Table (PDT) is used to specify the output format, control codes, and transfer of characters to a printer. When you use a PDT, the Windows based printer driver is not used. The PDT defines all information used to generate the print output. Click the **PDT File** button to select a compiled Printer Definition File.

To create a new PDT, you must first create a Printer Definition File (PDF), and then compile the PDF into a PDT. (If an uncompiled PDF file is selected, the compilation is performed automatically each time the print session is used. This feature has a performance overhead and is available mainly for ease of development). The PDF is a text file that defines macros and session parameters.

The PDT may override changes you make to **Font**, **Margins**, or **Page Setup**, although it is now possible to support these options in the PDT file. If you want to print in a particular font with a PDT, you should alter the startup sequence in the PDT to tell the printer which font you require. When a PDT file is used, Host Print service uses the definitions of NL, CR, FF, and LF from the PDT file to allow it to format the print job correctly.

Click **Printer Language is Hewlett-Packard (HP) PCL** if your PDT contains PCL commands. Limited support for Manager-defined font, margins and paper setup settings is automatically supported for HP (PCL), with full support available for customizing the Printer Definition File.

Font

To make changes to the font, click **Font**. Click the **Font** button. Make your selections in the Windows-based **Font** dialog box,and then click **OK**.

Use Fixed Font Size

Select the **Use Fixed Font Size** to select a user-defined font size. When this box is checked, the Host Print service will use the configured point size regardless of whether the data will fit (it will be clipped to fit the page if it is too large). When this option is selected, the Horizontal and Vertical scaling options on the **Advanced** tab are not available.

Print Session Properties: Page Layout

All these options are automatically available using the Windows GDI. Supporting these options using a PDT file may require extra configuration.

Horizontal Controls

Characters per line

The default is 132 characters per line. Type 80, 158, or another number for characters per line to change the default. The maximum line length is 255 characters.

Click **Override Host Commands** to force the specified settings and ignore host commands.

Vertical Controls

Lines per Page

Select **Lines Per Page** and then set the number of lines to be printed on each page.

Lines Per Inch

Select **Lines Per Inch** and then set either 6 or 8 lines per inch to set the default.

Click **Override Host Commands** to force the specified settings and ignore host commands.

Margin Settings

To change the margins, select **Margins**, and then click **Setup**. The **Page Setup** dialog box appears. Make your changes, and then click **OK**.

Page settings

Settings for Paper Source, Page Size, and Orientation are controlled from the Printer section. Click **Override Host Commands** to force the specified settings and ignore host commands.

Print Session Properties: Advanced

Filter DLL

Click **Filter DLL** to pass the printer data stream to a third-party or user-supplied DLL. Clicking **Filter** activates the **DLL File** button. The **Select Filter DLL** dialog box appears. Make your selection, then click **Open**.

Ignore Transparent Sections for PDT Formatting

Selecting this option will cause those sections of the print data stream that have been marked as transparent to be ignored when using a PDT file to format the data. Note, when using the GDI to format the data, transparent sections are automatically ignored. When such data is discarded, a log is written to the Event Log, containing the first 32 bytes of discarded data. This logging can be globally disabled from the Host Print service properties page.

Do Not Scale Horizontally

Selecting this option will turn off the horizontal scaling feature of the printer driver.

Do Not Scale Vertically

Selecting this option will turn off the vertical scaling feature of the printer driver.

3270 Printing

Transparency is ASCII

Click **Transparency is ASCII** to indicate that transparent data from the host is in ASCII and needs no translation from EBCDIC to ASCII. Selecting **Transparency is ASCII** causes the Windows-based Host Print service to not put the received data through an EBCDIC to ASCII translation table before printing.

Transparency Custom Byte

Transparency Custom Byte indicates the character designated to start a sequence of transparent data (the transparent data may or may not be ASCII). The IBM standard is 0x35, but if the host print job uses another value (for example 0x36, and so on), then this should be specified here.

No Line Formatting

Prevents the SNA Print Service from inserting its own Carriage Return/Line Feed (CR/LF) according to the dimensions specified in the Default Page Width field (also on this property page). As No Line Formatting is a special case, this box is usually not checked. It is a useful option when using physical printers that do their own wrapping or are told to do their own wrapping with an Esc sequence. The Esc sequence that causes a printer to do its own End-of-line wrap on PCL printers is <Esc>&s0C.

Printer Time Out

Default is 10 seconds. This parameter optimizes performance and system resources. If the printer is always available, system resources are consumed. However, if the printer is constantly being "opened" and "closed" with each print job sent, performance suffers. By setting the time out, the printer will remain available for jobs sent close together to improve performance, and yet "close" during non-busy times to free up resources.

Print Session Properties

The following tabs are available on the Print Session 3270 Properties sheet:

Print Session Properties: APPC

Remote APPC LU

Choose **Alias** or **Fully Qualified Name** to select another remote APPC LU.

You must supply either a **Remote APPC LU Alias** or a **Fully Qualified Name** before you can leave this page. If you want to use a **Fully Qualified Name**, you need to select a **Remote APPC LU Alias** first, then select a **Local LU Alias**. Then you must go back and change from the **Remote APPC LU Alias** to the **Remote APPC LU Fully Qualified Name**. If you select **Fully Qualified Name** first, the **Local LU Alias** does not get initialized.

Network Name + LU Name in NETNAME.LUNAME syntax (for example, APPN.NORTHB).

Local LU Alias

Select the local LU alias from the drop-down list.

Mode Name

The default mode name is QPCSUPP. Click the drop-down list arrow to make another selection. The choices are: **#INTERSC**, **BLANK**, **QPCSUPP**, **QSERVER**.

AS/400 Device Name

You must enter the name for the AS/400 printer device, which is a descriptive name that distinguishes different printers on the network.

System Type

Select the type of system you are printing from; either AS/400, System/36, AS/36.

AS/400

Msg Queue Name

Enter the qualified name of the message queue to which operational messages for this device are sent.

Msg Lib Name

Enter the name of the library in which the message queue is located.

System/36, AS/36

APPC Print Session Properties: Security

User ID

Type your **User ID**.

Password

Type your **Password**. Click tab to put the cursor in the **Confirm** password box.

Confirm Password

Type your password again and click **OK**.

Print Session Properties: Job Format

Job

Do Not Format Print Job allows print jobs that are formatted with host software to bypass the Windows-based printing format system. The Windows-based Host Print service treats all received data as transparent. All data is passed directly to the printer.

HPT

Displays the **Host Print Transform Properties** box, where you can designate the printer type and paper sources.

The Host Print Transform feature changes print data on the AS/400 to the ASCII format needed by a PC printer.

Select a manufacturer type and model number. If you cannot find the type and model number in the list you can type them

in the box. If you do not know the manufacturer type and model number, see your system administrator or see the help on the AS/400.

Important
You must enter an asterisk "*" before the manufacturer type and model number. Make sure there are no spaces between t he asterisk, manufacturer type and model number, for example: *HPIIISI, or *IBM3812.

Format Print Job

Print jobs from a host system can be formatted one of two ways:

GDI

The Windows Graphical Device Interface (GDI) is used to format the print job. SCS codes in the data are interpreted and represented, and the GDI automatically supports all the configurable options such as font, margins, duplex, paper-source etc. Transparent sections (for example, containing PCL escape sequences) are just ignored. When such data is discarded, a log is written to the Event Log, containing the first 32 bytes of discarded data. This logging can be globally disabled from the Host Print service properties page.

Selecting this option disables the Ignore **Transparent Sections for PDT Formatting** on the **Advanced** tab.

Note
If your print job has a mixture of SCS codes and PCL escapes, then you should use a PDT. The PDT should contain mappings f or the SCS code functionality, and the PCL escapes should pass directly to the printer without being mapped.

PDT

A Printer Definition Table (PDT) is used to specify the output format, control codes, and transfer of characters to a printer. When you use a PDT, the Windows based printer driver is not used. The PDT defines all information used to generate the print output. Click the **PDT File** button to select a compiled Printer Definition File.

To create a new PDT, you must first create a Printer Definition File (PDF), and then compile the PDF into a PDT. (If an uncompiled PDF file is selected, the compilation is performed automatically each time the print session is used. This feature has a performance overhead and is available mainly for ease of development). The PDF is a text file that defines macros and session parameters.

The PDT may override changes you make to **Font**, **Margins**, or **Page Setup**, although it is now possible to support these options in the PDT file. If you want to print in a particular font with a PDT, you should alter the startup sequence in the PDT to tell the printer which font you require. When a PDT file is used, Host Print service uses the definitions of NL, CR, FF, and LF from the PDT file to allow it to format the print job correctly.

Click **Printer Language is Hewlett-Packard (HP) PCL** if your PDT contains PCL commands. Limited support for Manager defined font, margins and paper setup settings is automatically supported for HP (PCL), with full support available for customizing the Printer Definition File.

Font

To make changes to the font, click **Font**. Click the **Font** button. Make your selections in the Windows-based **Font** dialog box,and then click **OK**.

Use Fixed Font Size

Select the **Use Fixed Font Size** to select a user defined font size. When this box is checked, the Host Print service will use the configured point size regardless of whether the data will fit (it will be clipped to fit the page if it is too large). When this option is selected, the Horizontal and Vertical scaling options on the **Advanced** tab are not available.

APPC Print Session Properties: Advanced

Filter DLL

Click **Filter DLL** to pass the printer data stream to a third-party or user-supplied DLL. Clicking **Filter** activates the **DLL File** button. The **Select Filter DLL** dialog box appears. Make your selection, then click **Open**.

Ignore Transparent Sections for PDT Formatting

Selecting this option will cause those sections of the print data stream that have been marked as Transparent to be ignored when using a PDT file to format the data. Note, when using the GDI to format the data, Transparent sections are automatically ignored. When such data is discarded, a log is written to the Event Log, containing the first 32 bytes of

discarded data. This logging can be globally disabled from the Host Print service properties page.

Do Not Scale Horizontally

Selecting this option will turn off the horizontal scaling feature of the printer driver.

Do Not Scale Vertically

Selecting this option will turn off the vertical scaling feature of the printer driver.

Print LU Properties: General

LU Number

Enter the LU Number.

LU Name

Enter the LU Name.

Connection

The connection for this LU is shown. The connection cannot be changed here.

Pool

If the LU has already been assigned to a pool, the pool name appears here.

Comment

Optionally, enter a comment of not more than 25 characters.

Use Compression

Selecting this option will compress the data stream and reduce the local area network traffic.

User Workstation Secured

Selecting this option allows only this workstation to access the Host LU.

TN3270 Properties: Settings

Idle Timeout

Specify time limits. If the session is inactive for this length of time, then TN3270 service disconnects the client computer.

Init Status Delay

Specify time limits. This is the delay between the time when TN3270 service connects to a host session and the time the TN3270 service starts updating the client computer screen. There are often a large number of startup messages when the TN3270 service first connects to a host session, and this option gives the user the opportunity not to receive them all.

Message Close Delay

Specify time limits. When TN3270 service forces a client computer to disconnect (for example, when the Host Integration Server session to the host has been lost), it sends the client computer an error message to be displayed on the screen. This value specifies the time between sending the message to the client computer and closing the socket with the client computer (which causes some client computers to clear the screen, and so erase the message).

Refresh Cycle Time

Specify time limits. This is the delay between updates of the status on the display.

Default RU Sizes - Inbound and Outbound

This controls the RU size (SNA message size) used by the TN3270 service for logon messages to and from the host. The minimum value for inbound or outbound RU size is 256 bytes. If the host application sends large logon screens, these values should be increased.

Certificate CN

The common name of the certificate used if TLS/SSL is enabled.

Print Service Properties

Click **No Event Log for Skipping Transparent Section** to prevent an entry in the Event Log every time print services skips a transparent section found while printing a host print job. Skipping transparent sections can be enabled from the **Advanced** tab of a print session properties page.

3270

Click **Flush Final FF** causes Host Print services to explicitly form feed the document at the end of a print job. This is not needed for most print jobs and configurations.

Click **Use Proportional Font Change** to prevent overlapping characters in documents containing nonfixed-type fonts printed through Host Print services. Most standard 3270 data streams are designated for fixed-type fonts.

Click **Ignore Characters 3F and Under** to cause the print service to ignore Hexadecimal characters 3F and below. In LU 1 data streams, this option causes hexadecimal characters below 40 to be replaced by spaces.

Click **Delay Print Start** to delay the start of the print job until printable data is received by Host Print services. In a few environments, hosts may poll the Host Print service for activity by sending "empty jobs". In such cases, not selecting this option may cause blank pages to be printed between actual print jobs.

LU 3 Only

Click **Do All FF** to force the printer driver to honor all form feed commands. (In most jobs, this is unnecessary and may lead to unwanted blank pages.)

Click **Always do NL** to insert a new line when the print services determines that the Maximum Print Position has been reached for a particular line of data. Selecting this option will prevent data from printing over an existing line.

Click **No Space After FF** to prevent print services from inserting a space character following a form feed.

Click **Support Embedded Printer Codes** to allow unmarked embedded characters, such as EBCDIC X'27' characters, to pass through print services to the printer.

LU 1 Only

Click **Use Fixed Tabs** to disable normal tab functionality (such as aligning tabs in columns) and interpret each tab as a fixed number of spaces. The number of spaces for a tab stop is based on the size of the first tab in the Set Horizontal Format SCS control code.

APPC

Set the **Activation Retry Limit** to the number of times print services will attempt to activate the APPC conversation following a terminated connection. The default value is –1 (infinite), meaning that attempts to activate the session will continue until it is successful or terminated manually.

Click **Infinite Retry Limit** to force the system to retry printing the job until it is successful.

Set the **Activation Retry Interval** to the number of seconds to wait before trying to print the job again. The default value is 10 seconds.

Configuring an IP-DLC Connection

As with other connections, configuration requires setting parameters on the Connection properties dialog. It is recommended that you read this section before beginning the configuration process, so you can gather all the necessary information before you start.

To configure an IP-DLC connection

1. If you have just created a new IP-DLC link service connection, the **Connection** properties dialog will automatically appear. If it does not, or if you are configuring an already-existing IP-DLC link service connection, right-click the IP-DLC link service connection in the results pane of the MCC snap-in, and click **Properties**.
2. Fill in the required parameters according to the tables below.

General Page

The **General** page is similar to that used for other connections in Microsoft® Host Integration Server.

Property	**Comments**
Name	Name of the connection.
Link service	This list displays all currently configured IP-DLC link services.
Comment	Descriptive comment (optional).
Activation	Specifies conditions under which connection is activated.
On server startup	Connection is activated on server startup.
On demand	Connection is activated on demand.
By administrator	Connection can only be activated by system administrator.
Allowed direction	Specifies the connection directions. For more information, see the note under Peer system later in this table.

Outgoing calls	Connection is outgoing.
Incoming calls	Connection is incoming.
Both directions	Connection can be both incoming and outgoing.
Remote end	Specifies which system is the remote end.
Host system	Selecting this option signifies that the connection will be used for Dependent LU Server (DLUS) traffic and may have dependent LUs. You cannot associate independent APPC LUs with an IP-DLC connection remote end type of host system.
Peer system	Selecting this option signifies that the connection will be used for independent APPC LU sessions. You cannot associate dependent 3270 LUs with an IP-DLC connection remote end type of peer system. Because peer system IP-DLC connections do not support DLUS traffic, selecting this option will disable all controls on the **Address** page and the IP-DLC page (shown later in this topic). Only one peer connection can be associated with an IP-DLC link service. Note: Selecting Peer system automatically sets Activation to **On Server Startup** and Allowed direction to **Both**.

Address Page

Use this page to configure DLUS properties for the connection. The DLUS on the mainframe system operates in conjunction with the Dependent LU Requester (DLUR) on the local Host Integration Server computer. Together they route dependent (for example, 3270) sessions across the APPN network using the IP-DLC link service.

If you selected the Remote end as Peer system on the **General** page, all controls on this page will be disabled.

Property	**Comments**
Primary DLUS	Fill in these required fields for the primary server with the appropriate names.
Network name	Maximum of eight characters of SNA type A string, representing the APPN network in which to locate the primary DLUS.
Backup DLUS	Fill in these fields for the backup server with the appropriate names. These fields are optional.
Preferred route	Optionally, enter the IP address, host name, or fully qualified name for the NNS as routing server through which to connect to the DLUS. Using a separate NNS may improve host performance by offloading the DLUS host computer from operating as an NNS directory server. The IP-DLC link service will attempt to establish the connection to the DLUS using the preferred route address.

System Identification Page

The **System Identification** page is similar to that used for other connections in Host Integration Server.

If you selected the Remote end as Peer system on the **General** page, all controls on this page will be disabled.

Property	Comments
Network name	Network name.
Control Point Name	Control point name.
Node ID	Required. Enter a hexadecimal value to uniquely identify the physical unit (PU) for the host connection. This value must match an existing remote LEN-style PU definition. Default value is 05D FFFFF. The node ID value must match the value configured on the host DLUS computer.
Link compression	Optional. Choose a setting from the list. Default value is **None**.
Remote node name	Specify information for the remote DLUS.
Node ID	Required. Enter a hexadecimal value to uniquely identify the physical unit (PU) for the host connection. This value must match an existing remote LEN-style PU definition for the DLUS. There is no default value.

IP-DLC Page

Use this page to optionally set parameters specific to the IP-DLC connection.

If you selected the Remote end as Peer system on the **General** page, all controls on this page will be disabled.

Property	Comments
Connection retry limits	Optionally, use these controls to specify retry limits for activating a connection.
None	Specifies no retries.
Limited	Specifies the number of retries, from 1 through 65534. Default value is 8.
Delay after retry	Specifies the wait period after a retry, from 1 through 327670 seconds. Value must be a multiple of five. Default value is 10 seconds.
DLUR retry limits	Optional. Use these controls to specify retry action in case the DLUR connection fails. Default setting is Infinite.
Infinite	Specifies an infinite number of retries.
None	Specifies no retries.
Limited	Specifies the number of retries, from 1 through 65534. Default value is 8.
Delay after retry	Specifies the wait period after each retry, from 1 through 65535 seconds. Default value is 10 seconds.

Configuring an IP-DLC Link Service

As with other link services, configuration requires setting parameters on the **Link Service Properties** dialog box. It is recommended that you read this section before beginning the configuration process, so you can gather all the necessary information before you start.

To configure an IP-DLC link service

1. If you have just created a new IP-DLC link service, the **IP-DLC Link Service Properties** dialog box will automatically appear. To reconfigure an existing IP-DLC link service, right-click the IP-DLC link service in the scope pane of the MCC snap-in, click **Properties**, and then click the **Configure** button to load the IP-DLC Link Service Properties dialog box.
2. Fill in the required parameters according to the following table.
3. Click **OK** to persist the settings to the configuration file and registry. The **Insert Link Service** dialog box appears.
4. Click **Complete Configuration of the IP-DLC Link Service**.

Property	**Comments**
Service name	Displays the name of the link service being configured. For a new link service, the name is predefined as SNAIP#, where # is the ordinal number of the link service.
Service title	Enter a user-friendly text description of up to 128 characters. This name will be displayed in the Service Control Manager. The default value is IP-DLC Link Service #N, where N is the ordinal number of the link service.
Primary NNS	Enter the IP address, host name, or fully qualified name of the primary network node server. There is no default value.
Backup NNS	Optionally, enter the IP address, host name, or fully qualified name of the backup network node server. There is no default value.
No Preferred NNS	Selecting this will allow the IP-DLC link service to use the first available network node server. If this option is not selected, the link service will always use the Primary NNS when it becomes available.
Local address	This set of parameters assigns the link service to a particular IP address on the local computer. Every link service must be associated with a local IP address or logical connection, and neither the IP address nor the logical connection can be used by another IP-DLC link service on the local computer.
Adapter address	Selecting this option button forces the link service to use the default IP address associated with the network adapter specified in the adjacent list. (The alternative is to associate with a static IP address.) The list displays all available network adapters currently configured. Physical adapters are listed first, followed by virtual adapters (that is, WAN Mini-ports).
Static IP address	Selecting this option button forces the link service to use the IP address specified in the adjacent list. The list displays all static IP addresses defined to the local computer. Enter the IP address, host name, or fully qualified name for the IP-DLC link service.
Local APPN node	This set of parameters configures the network identification of the APPN branch network node implemented by the IP-DLC link service. Note: No other IP-DLC link service within the APPN network can use both the same network name and control point name as this one. Individual names may be reused, but the pair may not.
Network name	Enter the APPN network name for the IP-DLC link service operating as a local APPN node across which the IP-DLC link service will communicate. This value must be text with a maximum of eight characters and comply with the APPN naming convention. There is no default value.
Control point name	Enter the APPN control point name for the IP-DLC link service operating as a local APPN node. This value must be text with a maximum of eight characters and comply with the APPN naming convention. There is no default value. The control point name must be unique for this computer on the APPN network.

Use Dynamic PU Definition	Selecting this option allows the IP-DLC link service to attempt to use the dynamically defined PU definitions on the host.
Node ID	In the first box, enter any valid 3-digit hexadecimal number except the reserved numbers of 000 and FFF. In the second box, enter any valid hexadecimal number except the reserved number 00000. There is no default value.
Associated LEN node	Select an SNA service operating as an APPN LEN node to associate with this IP-DLC link service. Default value is the first SNA service on the local computer.

Downstream Pool

Enter a **Name** and **Comment** for the pool.

LUA pool

Enter a **Name** and **Comment** for the pool.

Messaging Help

Microsoft MSMQ-MQSeries Bridge is an adaptable system that can be customized. You can set up MSMQ-MQSeries Bridge to operate on almost any Message Queuing (also known as MSMQ) or IBM MQSeries network configuration.

For additional information on the MSMQ-MQSeries Bridge user interface, see the following sections:

In This Section

General Tab

This tab displays the following information:

Path Name	**Network name of the MSMQ-MQSeries Bridge computer.**
Service	Name of the Microsoft MSMQ-MQSeries Bridge Service.
Version	Version of the Microsoft MSMQ-MQSeries Bridge Service.
Status	Status of the Microsoft MSMQ-MQSeries Bridge Service (running, paused, or stopped).

Advanced Tab

By default, MSMQ-MQSeries Bridge allocates one thread for each type of message pipe.

If the MSMQ-MQSeries Bridge is connected to more than one MQSeries Queue Manager (QM), you should allocate a larger number of threads. This improves performance, and if a pipe to one QM fails, it lets the other pipes continue running.

For each type of message pipe, specify the following:

Parameter	**Description**
Max Threads	The maximum number of threads (recommended one per MQSeries QM).
Refresh Queue Cache	The interval (in minutes) at which the message pipes check for Cache Timeout (see Setting Message pipe properties).
Support MSMQ to Bridge Encryption	Check this option to enable the Encryption feature from MSMQ to MSMQ-MQSeries Bridge.
Replace '.' with '-' in Queue Manager	Enables the MQSeries QM to address an Message Queuing computer that has a '-' in its name. This is necessary because MQSeries does not support the '-' character. When this box is checked, MSMQ-MQSeries Bridge replaces '.' from the remote QM name on MQSeries with '-'. You should only enable this option if there is a '-' in the Message Queuing computer name.

MQI Channels Tab

MSMQ-MQSeries Bridge accesses MQSeries through MQI channels defined in both MQSeries and MSMQ-MQSeries Bridge.

Note
Each MQI channel connects an MSMQ-MQSeries Bridge to an MQSeries Queue Manager. Ordinarily, you should define one MQI channel for each connected network or foreign site.

First, define the channels in the MSMQ-MQSeries Bridge properties. Later, you can export the definitions to MQSeries (see Exporting MQSeries definitions).

The currently defined channels are listed in the dialog box. Click **Add** for a new MQI channel, **Properties** to edit the settings for an existing channel, or **Remove** to delete a channel.

On the **General** tab of the Channel Properties window, specify the following options:

Parameter	Description
Channel Name	A legal MQSeries name for the MQI channel. For convenience, you can assign the same name as you assigned to the connected network representing MQSeries in Message Queuing, for example IBMNT_CN.
MQSeries Queue Manager	The MQSeries Queue Manager to which the channel connects.
Transport Type	TCP/IP or SNA LU 6.2 communication.

On the **Address** tab of the Channel Propertieswindow, specify the following parameters for TCP/IP:

Parameter	Description
IP Address, Port	Of the MQSeries listener.

For SNA LU 6.2, specify:

Parameter	Description
Side Information Record	The CPI-C Symbolic Name defined in Host Integration Server.

For more information on the TCP/IP or SNA LU 6.2 configuration, see the IBM MQSeries documentation.

On the **Security** tab of the Channel Properties window, you may specify:

Parameter	Description
MCA User	An existing or new MQSeries user name, for example FMQUSER1, under which the server side of the MQI channel runs.

In MQSeries, you should set the permissions of the MCA User, that is, the queues that MSMQ-MQSeries Bridge can address. If you do not specify an MCA User, the server side of the channel runs under the default user name, which is the value of the MQSeries SYSTEM.DEF.SVRCONN parameter.

General Tab - CN

On the **General** tab of the CN properties window, specify the following options:

Parameter	Description
MQSeries QM Name	Select from the list of MQSeries Queue Managers to which you have defined MQI channels.

Note
If the same MSMQ-MQSeries Bridge is on two connected networks, do not connect them both to the same QM.

Parameter	Description
Reply to QM Name	The default MSMQ QM to which MQSeries should return report or acknowledgment messages. Ordinarily, you would enter the name of the MSMQ-MQSeries Bridge computer, for example **MSBridge1**.

The MSMQ-MQSeries Bridge Manager exports your entry to MQSeries as a queue manager alias (see Exporting MQSeries definitions). If the Microsoft® Windows® computer name contains an invalid MQSeries character such as a hyphen (-), replace it with another character such as an underscore (_) in the Reply to QM Name box. If you want to receive acknowledgments by nontransactional instead of transactional message pipe, add a % sign to the name (for example, MSBridge1%).

MQSeries uses the alias to identify the transmission queue where it should send acknowledgments. You can redirect the acknowledgments by specifying a different alias. For example, if you want to receive acknowledgments on another computer where MSMQ-MQSeries Bridge is installed, enter the name of the computer and define the name as an alias in MQSeries. For additional information, refer to your IBM MQSeries documentation.

Parameter	Description
Startup	Enabled or disabled at MSMQ-MQSeries Bridge startup (to change the status afterwards see Starting, Stopping, or Pausing an Object).

General Tab - Message Pipe

The **General** tab of the **Message Pipe Properties** window displays the following:

Parameter	Description
Status	**Running**, **Paused**, **Pending**, **Recovering**, **Stopped**,or **Error** (This is read-only; to change the status see **Starting, Stopping, or Pausing an Object)**.
Startup	Enabled or disabled at MSMQ-MQSeries Bridge startup.

For the MQSeries>MSMQ Transactional and Nontransactional message pipes:

Parameter	Description
Transmission Queue Name	A unique MQSeries transmission queue name for the pipe. The default names are <CN name>.XMITQ for the transactional message pipe and <CN name>.XMITQ.HIGH for the nontransactional message pipe. You can specify different names if you want.

For example, if the MSMQ-MQSeries Bridge computer name is MSBridge1, the default transmission queue names are MSBridge1.XMITQ and MSBridge1.XMITQ.HIGH.

Batch Tab

To optimize performance, MSMQ-MQSeries Bridge batches messages together. Increasing the batch size may improve performance. For transactional message pipes, this may increase the quantity of retransmitted data after a communication failure.

To change the batch size, specify any combination of:

Parameter	Description
Max. Number of Messages	The maximum number of messages in a batch.
Max. Accumulated Size	The maximum size (in bytes) of a batch.
Max. Accumulated Time	The maximum time (in milliseconds) during which messages are batched.

Transmission begins as soon as there are messages to be sent. When any of the above limits is reached the message pipe checks that the batch was fully received on the destination side.

Cache Tab

To reduce the overhead of opening and closing a queue for each message, MSMQ-MQSeries Bridge caches queue handles and reuses them when several messages are sent to the same queue. Specify the following option:

Parameter	Description
Cache Expiry	The time (in minutes) after which MSMQ-MQSeries Bridge closes an unused queue handle.

The system checks for Cache Timeout only at the Cache Refresh Time (see "Setting MSMQ-MQSeries Bridge Properties"). Thus the maximum time that an unused queue handle remains open is the Cache Timeout + Cache Refresh Time.

To refresh the cache immediately (for example to release a queue that is needed by another application), see "Refreshing the cache."

Retry Tab

If a message pipe fails, MSMQ-MQSeries Bridge tries to restart it automatically. For both the Short and Long retry cycles, specify:

Parameter	Description
Count	Maximum number of retries.
Delay	Interval between retries.

For example, you might specify a Short cycle of 3 retries at 30-second intervals. If the connection is still not successful, the system continues in a Long cycle of, for example, 10 retries at 300-second intervals.

Note

If you connect by TCP/IP, set the Delay to at least twice the keep-alive time of the destination MQSeries computer. This enables the MQSeries listener to release the resources of a broken connection before MSMQ-MQSeries Bridge tries to reconnect.

Trace Utility Help

If you are working to improve performance or solve a problem in Host Integration Server , you can use trace tools to determine the source of the problem.

A trace file contains a record of internal activities. When you enable the Trace utility, the Trace files are stored in the TRACES folder of the Host Integration Server folder. Trace files have the extension .ATF.

When gathering information about system difficulties, it is best to start with event log information. Event log information is generally more straightforward to interpret than tracing information. However, if you call product support, the technician may ask for some or all of the following information:

- Tracing. Trace files record activity between or within components of Host Integration Server. Trace files provide detailed information about the exact sequence of events occurring within Host Integration Server or between Host Integration Server and another system on the network. Tracing is turned on in Host Integration Server using the SNA Trace tool, which is located in the Applications and Tools program group of Host Integration Server. Trace files are created in the TRACES folder of the root folder with the filename extension .ATF.

- Windows Server event logs. When gathering information about system difficulties, it is generally best to start with the event logs, which record system and application events, including errors. Event logs are more straightforward to interpret than tracing information. Use tracing information if the event logs do not provide enough detail.

The Trace utility is started from SNA Manager, or the Trace icon in the Host Integration Server Applications and Tools program group. You can run the Trace utility on each server in the subdomain. For specific instructions on running traces, see the Host Integration Server online Help.

After you start the Trace utility in Host Integration Server or on a Host Integration Server client, you can enable or disable trace options.

Use the topics in this section to navigate through the Trace Utility User Interface. Click a topic below for more information.

In This Section

Trace Items Tab

The **Trace Items** tab has several items that must be selected and configured for the Trace application to operate properly.

List of Trace Items

The following components can be configured for tracing.

- SNA applications
- SNA Manager Client
- SNA Manager Agent (MngAgent)
- DB2 Network Library

Trace Items - Properties

Properties displays the **Trace Properties Sheet** that contains an **Internal Trace**, **Message Trace**, **API Trace** or the **TN3270 Internal Trace** tab depending on the Trace Item that was selected.

From the **Internal Trace**, **Message Trace**, **API Trace** and **TN3270 Internal Trace** tabs, you can either select a single item or click **Set All**.

To set Server trace properties

1. On the **Tools** menu, click **Trace Initiator**.
2. Click a component to trace from the list of Trace items.
3. Click **Properties**.
4. Click the appropriate trace tab.
5. Click the appropriate options for the trace, and then click **Apply**.

Trace Items - Clear All Traces

Clear All Traces will clear all trace settings and stop any running traces. If you only want to clear a specific trace item, double-click the item or select the item and click **Properties**. Then click **Clear All**.

When you select **Clear all Traces**, you will be presented with a dialog box to verify that you want to clear all trace setting.

Trace Items - Purge All Trace Files

Purge All Trace Files will delete all trace files from the Traces folder that is designated in the **Trace File Directory** tab.

When you select **Purge All Trace Files**, you will be presented with a dialog box to verify that you want to delete all trace files. If you purge all trace files, the trace settings are not deleted.

Tracing Global Properties Tab

The **Tracing Global Properties** tab has several items that can be modified to adjust how Trace runs. These items include:

Trace File Flip Length

The default size is 20,000,000 bytes.

You can change the maximum length by highlighting the number and typing a new value.

Tracing Global Properties tab to set the length of the trace file

1. In the **Trace Settings** dialog box, click the **Tracing Global Properties** tab.
2. Enter a value for the **Trace File Flip Length** box, and then click **OK**.

Write Traces on a Background Thread

Check this box to run tracing in the background. If the box is cleared (blank), tracing runs in the foreground.

To reduce performance impacts caused by tracing Host Integration Server 2006 components, traces can be queued and written by a background thread when this box is checked. Otherwise, trace files will be written immediately.

Background Thread Priority

If you select **Write Traces on a Background Thread**, check only one item to set the level of priority for tracing to run within the Microsoft Windows operating system. Highest gives tracing the highest level of priority, which means that tracing takes precedence over other jobs. Idle means that tracing runs when the CPU is idle.

Trace File Directory Tab

The following information details the **Trace File Directory** tab.

The **Trace File Directory** tab allows you to change where the Trace Initiator files will be stored.

Use **Browse** or enter a new location.

3270 Client Help

The Microsoft Host Integration Server 3270 Client enables your personal computer (PC) to access IBM hosts on the Systems Network Architecture (SNA) network. With the Host Integration Server 3270 Client, you can establish a single logical unit (LU) for your SNA session. You can connect to the following host application environments:

- Virtual Machine/Conversational Monitor System (VM/CMS)
- Time-Sharing Options (TSO)
- Customer Information Control System (CICS)

Once connected to a host application, you can transfer information between the host system and your computer without using an intermediate storage medium. This gives you the ability to:

- Transfer a file back and forth between your computer and the host. You can then continue working with the file even if the host is unavailable or slow.
- Edit a file and return it to the host, where it can be shared by other users.
- Print the information in the active window of a host application file.
- Extract information from the host for further analysis on your computer.

In This Section

Defining Session Settings

Before you can connect to a host, you must define the settings you will need for your session. These settings include the LU or Pool Name and the Host Code Page.

To define session settings for a Host Integration Server connection

1. On the **Session** menu, click **Session Configuration**. The **3270 Settings** dialog box appears.
2. In the **LU or Pool Name** box, select the correct LU or pool name.
3. In the **Host Code Page** box, select the correct host code page. The default is United States.
4. Optionally, use a **Script File** to record and playback an automatic logon procedure. For more information, see Recording and Playing Logon Scripts.
5. Click **OK**.

To define session settings for a TN3270E connection

1. On the **Session** menu, click **Session Configuration**. The **3270 Settings** dialog box appears.
2. Enter the **Server Name or Address**. This should be the IP address of the Host Integration Server running the TN3270 service.
3. Select the **Model** to use for the connection from the drop down list.
4. Leave **Device** blank if you want to allow the TN3270 service to select an LU. Optionally, you can enter the name of a specific LU or LU pool.
5. Enter the **Port** number to use when connecting to the TN3270 service. The default is 23.
6. In the **Host Code Page** box, select the correct host code page. The default is United States.
7. Optionally, use a **Script File** to record and playback an automatic logon procedure. For more information, see Recording and Playing Logon Scripts.
8. Click **OK**.

Connecting to a Host Application

After defining session settings, you can automatically connect to the host every time you start the 3270 Client.

To connect to the host application

1. On the **Session** menu, click **Connect**. The **logon** dialog box appears, ready for you to enter your user ID and password.

To disconnect from the host application

1. On the **Session** menu, click **Disconnect**. The host disconnects.

To turn on the Autoconnect option

1. On the **Session** menu, click **Autoconnect**. A check mark will appear next to the option. When you start the Client, connection will be automatic.

To turn off the Autoconnect option

1. 1. On the **Session** menu, click **Autoconnect** to clear the check mark.

Numeric Override Facility

The numeric override facility allows a user to enter non-numeric characters in numeric-only fields.

To toggle numeric override on and off, press CTRL+N (the CTRL key and the N key simultaneously). When numeric override is on, the number 9 appears in the operator information area at the bottom of the terminal, indicating that non-numeric characters can be entered in numeric-only fields.

Note
Make sure that the host computer will accept non-numeric characters in numeric-only fields before using this facility.

How to Record and Play Logon Scripts

The 3270 logon scripts make it easy for users and administrators to automate the logon procedure. You can use a script provided by your system administrator, or create your own. You can run the logon script at any time. The logon script feature enables Single Sign-On to host applications for the 3270 Host Integration Server clients and 3270 weblets, provided your host security administrator has enabled account password synchronization.

To use a logon script provided by your system administrator

1. Start the 3270 Client.
2. On the **Session** menu, click **Connect**. The **logon** dialog box appears, ready for you to enter your user ID and password.
3. On the **Script** menu, click **Play**. The script runs, and you are logged on.

To create your own logon script

1. Start the 3270 Client.
2. On the **Session** menu, click **Connect**. The **logon** dialog box appears, ready for you to enter your user ID and password.
3. On the **Script** menu, click **Record**.
4. Logon to the host using the 3270 screens and keyboard. The **Record** facility converts these actions into a logon script. (Click here to see a Sample Logon Script.)

When you are finished recording your logon script, on the **Script** menu, click **Stop**.

1. To run your logon script, on the **Script** menu, click **Play**.

The scripts can be modified using any text editor, such as Notepad.

To run the logon script automatically each time you connect

1. On the **Script** menu, select **Auto Run**. When you establish a connection to the host, the logon script runs automatically.
2. Optionally, on the **Session** menu, select **Autoconnect**. Every time you start the 3270 Client, the connection to the host will be made and you will be logged on automatically.

To choose different script files

1. On the **Session** menu, select Session Configuration. The **3270 Settings** dialog box appears.
2. On the **Script File** box, click **Browse** and select another script file.
3. Double-click the script file you want to use.
4. Click **OK**.

See Also
Concepts
Sample Logon Script

Sample Logon Script

```
SETTIMEOUT 30,EXIT
WAITSESSION SSCP
WAIT 3
SEND vm1@E
WAITSESSION LULU
WAIT 3
SEND myuser@Tmypass@E
WAIT 3
SEND run myapp@E
WAIT 3
EXIT:
```

See Also

Tasks

How to Record and Play Logon Scripts

Defining File Transfer Settings

Using the 3270 Client, you can transfer a file between your computer and a host application. However, before you can send a file to the host or receive a file from the host, you must define the transfer settings that you need. These settings tell the 3270 Client which host environment you will be using (VM/CMS, TSO, or CICS) as well as the parameters for the connection between your computer and the host.

Before starting this procedure, review the configuration parameters under step 3 below to ensure that you have the necessary information at hand.

To define file transfer settings

1. On the **Transfer** menu, click **Settings**. The File Transfer Settings dialog box appears.

2. Under **Host Operating System**, select the option for the correct host environment. The host environment option that you select determines the rest of the settings you enter in the File Transfer Settings dialog box. Settings that you do not need for your host environment are unavailable.

3. Find the items that apply to your configuration, and supply the information as follows:

   Host Program Name Specify the host program name by typing the host name for the program used by the 3270 Client.

   Timeout Specify the amount of time, in seconds, that the 3270 Client will wait for a host response to the file transfer request.

   Packet Size Specify the packet size transferred at one time. The larger the size, the faster the file is transferred.

   Block Size (TSO) Select the Block Size check box, and specify the size of the data blocks, in bytes, in the new data set on your TSO volume.

   Issue Clear (VM/CMS and CICS) Select this check box to request the host to clear the terminal screen before transferring files.

   Host Code Page Select the correct EBCDIC code page translation table used by the host.

   PC Code Page Select the correct ASCII code page translation table used by your computer.

   Record (VM/CMS and TSO) Specify the type for the logical record length by selecting the correct option button for your configuration. This parameter controls whether the logical record length—the number of characters in each record of a file transferred to the host—is fixed, variable, undefined, or default; that is, controlled by host defaults (TSO only). A transferred file that replaces or appends to an existing file automatically takes the logical record length of the existing file.

   To update an existing host file, select Default.

   To create a new host file, select the record length type: Fixed, Variable, or Undefined.

   If you select Fixed, specify the logical record length in the Fixed box.

   Space (TSO) Select this check box to specify the space for a new data set, then select the option that corresponds to the unit of measurement (Blocks, Tracks, or Cylinders) and type the number of units to allocate. To allocate additional space, in the second box, type the number of additional blocks, tracks, or cylinders to allocate. To find out the correct amount of space to specify, contact the host administrator.

4. Click OK.

Specifying a File

Once you have defined the settings for transferring files, you can send a file to the host or receive a file from the host. You can select a file to transfer in one of two ways: by typing the name of the specific file to send or receive, or by browsing through a list of available files.

To type the name of a specific file

1. On the **Transfer** menu, click **Send File** or **Receive File**. Either the Send File dialog box or Receive File dialog box appears.

2. In the **PC File Name** box, type the name of the file to send or receive. Note that when receiving a file from the host application, if you type the name of an existing PC file, it will be overwritten by the host file.

3. To review your settings, click **Settings**.

To browse through a list of available files

1. On the **Transfer** menu, click **Send File** or **Receive File**. Either the Send File dialog box or Receive File dialog box appears.

2. Click **Browse**. The **Open** dialog box appears.

3. In the **File Name** list, select the file you want to send or receive. To see files of another type, select the type from the **List Files of Type** box.

4. Click **OK**. The Send File or Receive File dialog box reappears with your selection in the PC File Name box.

5. To review your settings, click **Settings**; to continue.

Sending or Receiving a File

Once you have defined the settings you need to transfer files, and selected the file to transfer, you can send a file to or receive a file from the host. To send or receive a file, you must supply certain configuration information. This information includes the host filename, host file type, host file mode, data set name, carriage return, whether you are appending to an existing file, EBCDIC conversion, and carriage return line feed.

To send a file to the host or receive a file from the host

1. After specifying the file, find the items that apply to your configuration, and supply the information as follows:

   Specify the host file type. For CICS, select either ASCII File or Binary File.

   Host File Mode (VM/CMS) Specify the host file mode, the specific minidisk where the user data resides. To find out the mode for a specific file, from an Client session, type the following command, substituting the name of the file for filename: list filename * *.

   In the resulting display, the third column contains the file mode. The default host file mode is A1.

   Data Set Name (TSO) Specify the host data set name. Be sure to include the Member name, in parentheses, if you are sending to a host partitioned data set (PDS). The following is an example of a data set name for a PDSDEF_009:

   PROJECT.GROUP.TYPE(MEMBER)

   Carriage Return (CICS) Select the carriage return option for the file you are sending or receiving. Delete CR/LF removes all carriage return/line feed characters from the file sent to the host. No CR/LF leaves the carriage return/line feed characters in the file when transferring to the host. None of the Above uses the default implied by the type of file (ASCII or binary) being transferred. The default for ASCII (being converted to EBCDIC) is CR/LF. The default for binary is No CR/LF, and CRLFs will not be interpreted.

   Append to Existing File (send) Select this check box to add new data to the end of an existing host file.

   Convert to EBCDIC (send) Select this check box to convert from ASCII to EBCDIC.

   Convert from EBCDIC (receive) Select this check box to convert from EBCDIC to ASCII. Use this option for text files rather than binary files.

   Delete Carriage Return/Line Feed CR/LF (send) Select this check box to replace all CR/LF characters with end of record marks.

   Insert Carriage Return/Line Feed CR/LF (receive) Select this check box to replace all end-of-record marks with CR/LF characters. Use this option for text files rather than binary files.

2. Click **OK**. The 3270 Client begins the transfer.

Using Macros

The 3270 Client provides macros for ease in using host function keys.

To use a macro, select the correct macro from the Macro menu. The appropriate function key is sent to the host.

The following table shows the macros for the 3270 Client:

Macro	Description
Attention	Sends an Attention key to the host. The Attention key is used to temporarily stop the present display station activity.
Clear	Sends a Clear key to the host. This macro clears the display screen.
Reset	Sends a Reset key to the host. Reset unlocks the keyboard and clears operator errors.
Sys Request	Sends a Sys Req key to the host. With most systems, use this macro to notify the host system that the display station is ready to select a new program or activity.

Printing the Screen

After you have established a connection to the host, you can print the information in the active window.

To print the screen

- On the **File** menu, click **Print**.

To change printer options

1. On the **File** menu, click **Printer Setup**.

   The Print Setup dialog box appears.

2. To change the printer, from the **Printer** list, select the correct printer.

   To view **Advanced Document Properties**, click **More**.

   To view printers on the network, click **Network**.

3. Change any other printer options as necessary.

4. Click **OK**.

Copying and Pasting Displayed Information

The 3270 Client allows you to make a copy of the information in the display, so that you can use that information in other applications. To do this, you select a portion of the displayed information, and then copy it to the Windows Clipboard. After copying information to the Clipboard, you can change to a different application and transfer the information into that application.

In reverse fashion, you can copy information from an application other than the Client, then paste the information into the Client. If the emulation session is in a state in which information can be received (indicated by the status line), the pasted information will be entered as a series of keystrokes.

To copy and paste displayed information

1. Select the portion of the displayed information that you want to copy

   -or-

   on the **Edit** menu, click **Select All**.

2. On the **Edit** menu, click **Copy**. The selection is copied to the Clipboard.

3. Change to the program in which you will insert the contents of the Clipboard.

4. Position the cursor at the location where you want to insert the contents of the Clipboard.

5. On the **Edit** menu, click **Paste**.

To copy information from an application and paste it into the Client window

1. From the application, copy information to the Clipboard by using the application's standard copy command(s).

2. Change to an active Client session, or start the Client and begin a session.

3. Check to see that the session is in a state in which keystrokes can be received.

4. On the **Edit** menu, click **Paste**. The copied information is entered as a series of keystrokes.

Saving a Configuration

After you have configured a connection to the host, you can save it. While you are connected to the host application, you cannot open a new or existing configuration.

To save a configuration

1. On the **File** menu, click **Save**.The Save As dialog box appears.
2. In the **Directories** box, select the appropriate directory.
3. In the **File Name** box, type a name for the configuration file or select a file from the list.
4. Click **OK**. Now, when you click **Open** on the **File** menu, the configuration appears as an available configuration.

To open a configuration

1. On the **File** menu, click **Open**. The Open dialog box appears.
2. In the **Directories** box, select the appropriate directory. To open a file from a directory on the network, click **Network**.
3. In the **File Name** box, type a name for the configuration file or select a file from the list.
4. Click **OK**.

To create a new configuration

- On the **File** menu, click **New**. The configuration settings are reset to their default values.

To rename a configuration

1. On the **File** menu, click **Save As**. The Save As dialog box appears.
2. In the **Directories** box, select the appropriate directory. To save a file to a directory on the network, click **Network**.
3. In the **File Name** box, type a new name for the configuration file or select a file from the list.
4. Click **OK**. Now, when you click Open on the File menu, the configuration is renamed and appears as an available configuration.

3270 Client Standard Keyboard Map

The following table maps values from the 3270 keyboard to the Enhanced 101 keyboard.

3270 Keyboard	**Enhanced 101 Keyboard**
DUP	CTRL ALT INSERT
FIELD MARK	SHIFT HOME
ERASE TO END OF FIELD	SHIFT DELETE
ERASE INPUT	ALT END
RESET	CTRL + R
HOME	HOME
TAB	TAB
BACKTAB	SHIFT TAB
ENTER	ENTER
NEWLINE	SHIFT ENTER
FAST CURSOR LEFT	NOT SUPPORTED BY THIS EMULATION
FAST CURSOR RIGHT	NOT SUPPORTED BY THIS EMULATION
CURSOR UP	UP ARROW
CURSOR DOWN	DOWN ARROW
CURSOR LEFT	LEFT ARROW
CURSOR RIGHT	RIGHT ARROW
INSERT	INSERT
DELETE	DELETE
ATTENTION	ESC
SYSTEM REQUEST	CTRL+ S
CLEAR	PAUSE
PA1	ALT INSERT
PA2	ALT HOME
PA3	SHIFT PAGE UP

CurSel	ALT F3
PF1	F1
PF2	F2
PF3	F3
PF4	F4
PF5	F5
PF6	F6
PF7	F7
PF8	F8
PF9	F9
PF10	F10
PF11	F11
PF12	F12
PF13	SHIFT F1
PF14	SHIFT F2
PF15	SHIFT F3
PF16	SHIFT F4
PF17	SHIFT F5
PF18	SHIFT F6
PF19	SHIFT F7
PF20	SHIFT F8
PF21	SHIFT F9
PF22	SHIFT F10
PF23	SHIFT F11
PF24	SHIFT F12

OIA Inidcators

The Operator Information Area (OIA) is used to display the status of a 3270 screen session. The OIA is the last (bottom) line display in the terminal window of the 3270 session. Except for the initial three positions (system connection indicators), status line indicators only appear when they apply.

The following tables describe the OIA and explain each indicator. In the table, position refers to the column(s) that a specific indicator occupies on the OIA, where the left-most column of the OIA is position 1. Note that the "female" symbol used for certain system connection and input-inhibited indicators cannot be displayed in this table. In its place is the "^" symbol.

Following is a list of symbols found on each system connection and their respective meanings.

System Connection (position 1)

Does not apply to Host Integration Server 3270 Client.

System Connection (position 2)

B Connection uses an SNA protocol.

A Connection uses a non-SNA protocol.

System Connection (position 3)

b Connected to a host application (LU-LU session).

^ Connected to host system services (SSCP-LU session).

? Session is established, but is not connected to any host program (unknown).

Session ID (positions 4-7)

aaaa First four characters of your configured long name session ID.

Input-Inhibited messages (keyboard locked) (positions 9-17)

X COMMnnn Communications error. See Communications Status (positions 19-26).

X MACHnnn Machine check error.

X PROGnnn Program check error. See Communications Status (positions 19-26).

X SYSTEM Keyboard input is being processed by the host. Wait for the message to clear or press RESET.

X[] The host requires more time to process your request. Wait for indicator to clear or press RESET.

X -f Function is not supported. Press RESET and select a valid function.

X^> More data was typed than the field allows or can accept. Press RESET and re-enter the data.

X <-^-> Protected field; you cannot type data. Press RESET and move the cursor to an unprotected field.

X -f^X Function is not authorized. Press RESET and select a valid function.

X^+? Invalid accent/character combination typed. The accent is displayed in position 12. Press RESET and type a valid combination.

Communications Status (positions 19-26)

COMM500 All the link services configured for use by the connection were inactive.

COMM504 Communications are not established with the host. This message disappears as soon as communication is established with the host. If this message does not disappear, check to ensure proper settings of the host network address, local node ID, or local SAP.

COMM505 The host terminated the connection used by your 3270 session.

COMM510 The host has deactivated the communications link. Report the message to your system administrator.

COMM518 A segment was out of sequence. Contact your system administrator.

COMM695 A communications error occurred.

COMM5nn The communications link is still being established; the keyboard is locked. Wait for the message to clear; also check

Input-Inhibited messages.

PROG703 A session control, data flow control or network control command is not supported. Press RESET, and try the operation again.

PROG705 A sequence number error has occurred. Press RESET, and try the operation again.

PROG706 A chaining error has occurred. Press RESET, and try the operation again.

PROG707 A bracket state error has occurred. Press RESET, and try the operation again.

PROG708 A data traffic reset state exists. Press RESET and try the operation again.

PROG717 The support level requested is not valid. Press RESET, and try the operation again.

PROG723 The LU type requested is not valid. Press RESET, and try the operation again.

PROG724 The screen size requested is not valid. Press RESET, and try the operation again.

Screen Session LU Address (positions 79-80)

Nn The LU address of the currently displayed 3270 screen session.

5250 Client Help

The Microsoft Host Integration Server 5250 Client is a powerful communications product that enables you to access AS/400 systems on the Systems Network Architecture (SNA) network from your computer. Once connected to a host application, you can print the screen, copy and paste displayed information via the Clipboard, and save a configuration.

This section contains:

Defining Session Settings

Before you can connect to an AS/400 system, you must define the settings you will need for your 5250 session such as the Local LU Alias, Remote APPC LU Alias, Host Code Page, and Display Size. You can also change the Device Name, although this is generally not necessary.

To define basic session settings for the 5250 Client for a Host Integration Server

1. On the **Session** menu, click **Session Configuration**. The 5250 Session Information dialog box appears.
2. Select the Server Type: **Host Integration Server**.
3. In the **Local APPC LU Alias** box, type the Local LU Alias provided by your system administrator. In some cases, the Local LU Alias provided by your system administrator may be the same as your user name. In other cases, your system administrator may instruct you to leave the box blank; this allows the Host Integration Server to use the default local LU configured for your user name.
4. In the **Remote APPC LU Alias** box, select the system name of the AS/400, as provided by your system administrator. Your system administrator may instruct you to leave the box blank; this allows the Host Integration Server to use the default system name (also called the remote LU) configured for your user name.
5. In the **Host Code Page** box, select the correct host code page. The default is English-US.
6. Under **Display Size**, select the option button for the correct display size. The default is 24 x 80.
7. To specify a device name other than the default, in the **Device Name** box, type the correct device name.
8. Select the **System 36 Compatibility** check box to connect to a System 36 computer.
9. Select the **Automatic Logon** check box to use the single sign-on feature. You must install the Host Security Integration feature to enable single sign-on.
10. Click **OK**.

To define basic session settings for the 5250 Client for a TN 5250 Server

1. On the **Session** menu, click **Session Configuration**. The 5250 Session Information dialog box appears.
2. Select the Server Type: **TN 5250**.
3. Enter the Server Name or IP Address for the server where Host Integration Server is installed. If you are unsure, see your system administrator.
4. Select a **Terminal Type** from the list box to allow the TN5250 Service to accept client requests from TN5250 clients emulating those types of terminals. Clear a name to cause TN5250 Service to reject requests from clients emulating that terminal type. All terminal names are selected by default.
5. Select **Use Default** to use the port number configured with the service (on the **Host Integration Server Service Properties** page).
6. You can override the default value for a given session by selecting Use and typing another port number.

   Note that TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client to connect to multiple host computers. See the Host Integration Server help for more information.

7. In the **Host Code Page** box, select the correct host code page. The default is English-US.

8. Click **OK**.

Using the Keypad Menu

Standard 5250 function keys can be selected from the Keypad menu. The direct keyboard mapping is displayed in the right-most column of the menu.

To send a 5250 keystroke to the host

- From the Keypad menu, choose the correct 5250 function. The appropriate function key is sent to the host.

Note
Limited keyboard remapping is provided for the standard IBM 101 keyboard.

Printing the Screen

After you have established a connection to the host, you can print the information in the active window.

To print the screen

- On the **File** menu, click **Print**.

To change printer options

1. On the **File** menu, click **Printer Setup**. The Print Setup dialog box appears.
2. To change the printer, in the **Printer** list, select the correct printer. To view Advanced Document Properties, click **More**. To view printers on the network, click **Network**.
3. Change any other printer options as necessary.
4. Click **OK**.

Copying and Pasting Displayed Information

The 5250 Client allows you to make a copy of the information in the display, so that you can use that information in other applications. To do this, select a portion of the displayed information, and then copy it to the Windows Clipboard. After copying information to the Clipboard, you can change to a different application and transfer the information into that application.

In reverse fashion, you can copy information from an application other than the Client, and then paste the information into the Client. If the emulation session is in a state in which information can be received (indicated by the status line), the pasted information will be entered as a series of keystrokes.

To copy and paste displayed information in the Client window

1. Select the portion of the displayed information that you want to copy

   -or-

   on the **Edit** menu, click **Select All**.

2. On the **Edit** menu, click **Copy**. The selection is copied to the Clipboard.

3. Change to the application in which you will insert the contents of the Clipboard.

4. Position the cursor where you want to insert the contents of the Clipboard.

5. On the **Edit** menu, click **Paste**.

To copy information from an application and paste it into the Client window

1. From the application, copy information to the Clipboard by using the applications standard copy command(s).

2. Change to an active Client session, or start the Client and begin a session.

3. Check to see that the session is in a state in which keystrokes can be received.

4. On the **Edit** menu, click **Paste**. The copied information is entered as a series of keystrokes.

Saving a Configuration

After you have configured a connection to the host, you can save it. While you are connected to the host application, you cannot open a new or existing configuration.

To save a new configuration

1. On the **File** menu, click **Save**. The Save As dialog box appears.
2. In the **Directories** box, select the appropriate directory.
3. In the **File Name** box, type a name for the configuration file or select a file from the list. To save the configuration to a directory on the network, click **Network**.
4. Click **OK**. Now, when you choose **Open** from the **File** menu, the configuration appears as an available configuration.

To open a configuration

1. On the **File** menu, click **Open**. The Open dialog box appears.
2. In the **Directories** box, select the appropriate directory. To open a file from a directory on the network, click **Network**.
3. In the **File Name** box, type a name for the configuration file or select a file from the list.
4. Click **OK**.

To create a new configuration

- On the **File** menu, click **New**. The configuration settings are reset to their default values.

To rename a configuration

1. On the **File** menu, click **Save As**. The Save As dialog box appears.
2. In the **Directories** box, select the appropriate directory. To save a file to a directory on the network, click **Network**.
3. In the **File Name** box, type a new name for the configuration file or select a file from the list.
4. Click **OK**. Now, when you choose **Open** from the **File** menu, the configuration is renamed and appears as an available configuration.

5250 Client Standard Keyboard Map

The following table maps values from the 5250 keyboard to the Enhanced 101 keyboard.

5250 Keyboard	Enhanced 101 Keyboard
F1	F1
F2	F2
F3	F3
F4	F4
F5	F5
F6	F6
F7	F7
F8	F8
F9	F9
F10	F10
F11	F11
F12	F12
F13	SHIFT+F1
F14	SHIFT+F2
F15	SHIFT+F3
F16	SHIFT+F4
F17	SHIFT+F5
F18	SHIFT+F6
F19	SHIFT+F7
F20	SHIFT+F8
F21	SHIFT+F9
F22	SHIFT+F10
F23	SHIFT+F11
F24	SHIFT+F12

Alternate Cursor	ALT+F9
ATTENTION	ESC
Backspace	BACKSPACE SHIFT+BACKSPACE
BACKTAB	SHIFT+TAB
CLEAR	PAUSE SHIFT+PAUSE
Cmd	SHIFT+ALT(left) SHIFT+ALT(right)
CURSOR UP	UP ARROW 8 (num pad) SHIFT+UP ARROW
CURSOR DOWN	DOWN ARROW 2 (num pad) SHIFT+DOWN ARROW
CURSOR LEFT	LEFT ARROW 4 (num pad) SHIFT+LEFT ARROW
CURSOR RIGHT	RIGHT ARROW 6 (num pad) SHIFT+RIGHT ARROW
DELETE	DELETE . (num pad)
DUP	SHIFT+INSERT
END	END 1 (num pad) SHIFT+END
ENTER	ENTER
ERASE EOF	NA
ERASE EOL	ALT+HOME
ERASE INPUT	ALT+END

FAST CURSOR LEFT	ALT+LEFT ARROW
FAST CURSOR RIGHT	ALT+RIGHT ARROW
FIELD EXIT	+ (num pad)
FIELD MARK	NA
FIELD -	- (num pad) SHIFT+-(num pad)
FIELD +	+ (num pad)
HOME	HOME 7 (num pad) SHIFT+HOME
HELP	SCROLL LOCK SHIFT+SCROLL LOCK
HEX	ALT+F7
INSERT	INSERT 0 (num pad)
LOCAL HELP	CTRL+SHIFT+H
LOCAL HELP CURSOR	CTRL+H
LOCAL PRINT	CTRL+P
NEWLINE	SHIFT+ENTER
PAGE UP	PAGE UP 9 (num pad) SHIFT+PAGE UP
PAGE DOWN	PAGE DOWN 3 (num pad) SHIFT+PAGE DOWN
PA1	NA
PA2	NA
PA3	NA
PRINT	PRINTSCREEN

RESET	CTRL+R
SYSTEM REQUEST	CTRL+S ALT+NUM (*)
TAB	TAB SHIFT+NUM(+)
TEST REQUEST	ALT+PAUSE

Remapping for the Standard IBM 101 Keyboard Layout

When using the Host Integration Server 5250 Client, function keys located on the numeric keypad (such as PGUP, PGDN and ENTER) are not mapped to 5250 functions for PAGE UP, PAGE DOWN, and ENTER. The standard cursor control keys for PageUp and PageDown (located on the keypad positioned between the BACKSPACE and NUMLOCK keys) and the standard ENTER key do work.

The <Snaroot>\System\5250.kbd file can be edited to support limited keyboard remapping, for the standard IBM 101 keyboard layout. For example, the procedure below describes how to add support for the ENTER, PGUP and PGDN keys located on the numeric keypad:

To add support for the ENTER, PGUP and PGDN keys

1. Save the original copy of <Snaroot>\System\5250.kbd to 5250.old.

2. Add the following lines to the 5250.kbd file:

    - NumPgUp : PAGE_UP
    - NumPgDn : PAGE_DOWN
    - NumEnter: ENTER

Note
5250.kbd will already include default entries for the 5250 PAGE_UP, PAGE_DOWN, and ENTER functions, mapped to the PAGEUP, PAGEDOWN, and ENTER keys respectively.

1. Copy the 5250.kbd file to Keyboard.map, and place Keyboard.map in the same directory where the Win5250.exe program resides:

    - for Windows 3.x clients, the default is <sna.win>
    - for Windows 95 clients, the default is <sna95>\system
    - for Windows NT clients, the default is <sna>\system

2. . Restart the 5250 Client.

Note
The Host Integration Server 5250 Client does support limited keyboard remapping by following the instructions in the <Snaroot>\System\5250.kbd file. This keyboard mapping support is not available in SNA Server 2.x versions of the 5250 Client.

5250 Client Status Line

The following list describes the information displayed in the status line of the 5250 Client.

Connection Indicator The session is in the indicated state: connected or disconnected. When menu items are highlighted, an explanation of the menu item is displayed.

Status Indicators These indicators are only true when red.

SA (System Available) The AS/400 is operating and is available to the PC.

MW (Message Waiting) The AS/400 has one or more messages for you.

KS (Keyboard Shift) The keyboard is in shifted mode.

IM (Insert Mode) Characters inserted into an existing field will not type over the existing data.

II (Input Inhibited) Keyboard input is not being accepted by the AS/400. Try pressing the ERROR RESET key. If it is still highlighted, the system is processing your request.

KB (Keystroke Buffering) Keystrokes are being saved in a temporary buffer (storage space), because keyboard input is not being accepted by the AS/400 (as indicated by the II indicator). After the II indicator goes off, the keystrokes will be processed. To clear the keystroke buffering, press the ERROR RESET key.

System and Device Indicators These indicators appear only when the Client is connected to the AS/400.

Administrator's Reference

The Administrator's Reference section provides technical information about working with Microsoft Host Integration Server.

To use this reference section effectively, you should be familiar with the following:

- Host Integration Server
- Microsoft Windows
- IBM Systems Network Architecture (SNA) concepts

In This Section

Document Conventions explains the formatting conventions this documentation set uses to distinguish important elements of text.

Common Acronyms and Abbreviations lists common industry abbreviations and acronyms, along with their meanings.

Host Integration Server Support explains how Host Integration Server works within Systems Network Architecture (SNA) networks.

Network Management describes the support that Host Integration Server provides for centralized network management

Network Protocols and Client/Server Communication describes how clients using different local area network (LAN) protocols can communicate with Host Integration Server computers.

Error Messages describes the audit and error messages database and how to access it.

Command-Line Interface describes the command-line interface that can be used when you want to view a configuration setting quickly, or when you wnat to store configuration commands.

Sample Host Definitions presents samples for establishing Host Integration Server connections to an IBM host system.

CICS and VTAM Sample Definitions for LU 6.2 presents sample definitions for CICS and VTAM for Host Integration Server connections to an IBM host system.

Sense Codes lists sense codes used by Host Integration Server.

Character Tables provides tables of the ASCII, ANSI, and IBM Extended character sets. It also provides a character translation table that can be used by the custom code page option of Host Print service.

For more information about systems network architecture, see the latest version of the IBM publication, *Systems Network Architecture Technical Overview* (GC30-3073).

For frequently used terms, see the Glossary.

Document Conventions

This *Administrator's Reference* uses the following formatting conventions to distinguish important elements of text.

Convention	Purpose
bold	Indicates syntax items that must be typed exactly as shown.
italic	Indicates variables.
UPPERCASE	Represents file names, paths, volumes, array and structure names, values, and conversation states. Also represents return values and key names (such as ALT or CTRL).
[brackets]	Enclose optional items in square brackets.
{braces}	Enclose required items in curly brackets.
| (vertical bar)	Stands for OR and separates items in syntax statements.
... (ellipsis)	Indicates that the preceding syntax item can be repeated. If a punctuation mark precedes the ellipsis, you must include the punctuation when repeating the item.
X'*nnnn*'	Indicates hexadecimal digits.

Common Acronyms and Abbreviations

The following table contains a list of common abbreviations used in this reference. For definitions of these and other terms, see the Glossary.

Abbreviation	Description
ACTLU	Activate Logical Unit
ACTPU	Activate Physical Unit
ADO	ActiveX Data Objects
API	application programming interface
APPC	Advanced Program-to-Program Communications
APPN	Advanced Peer-to-Peer Networking
ASCII	American Standard Code for Information Interchange
BB RQE	Begin Bracket, Exception Response
BTU	basic transmission unit
CD	Change Direction
CEB	Conditional End Bracket
CICS	Customer Information Control System
CCSID	Coded Character Set ID
COM	Component Object Model
CPI-C	Common Programming Interface for Communications
CSV	Common Service Verb
CTS	clear-to-send
DACTLU	Deactivate Logical Unit
DAF	destination address field
DCA	Digital Communications Associates, Inc.
DCOM	Distributed Component Object Model
DDM	Distributed Data Management
DDS	Digital Data System
DFC	data flow control

DLC	data link control
DLL	dynamic-link library
DLS	Distributed Link Service
DM	disconnect mode
DMA	Direct Memory Access
DMOD	Dynamic Module
DRDA	Distributed Relational Database Architecture
DSC	data stream compatible
DSR	Data Set Ready
DTR	Data Terminal Ready
EB	End Bracket
EBCDIC	Extended Binary Coded Decimal Interchange Code
EN	End Node
ESCON	Enterprise System Connection
FDDI	Fiber Distributed Data Interface
FEP	front-end processor
FM	function management
FRMR	frame reject
FTP	File Transfer Protocol
IMS	Information Management System
KB	Kilobyte
LAN	local area network
LCS	link connection subsystem
LL	record length field
LLC	Logical Link Control
LU	logical unit
LUA	Logical Unit for Applications (programming interface)

MAC	media access control
MB	Megabyte
MMC	Microsoft Management Console
MVS	Multiple Virtual Storage
NAU	Network Addressable Unit
NCP	Network Control Panel
NMVT	network management vector transport
NN	Network Node
N(r)	number returned
NRM	Normal Response Mode
NRZ	non-return to zero
NRZI	non-return to zero inverted
N(s)	number sent
ODBC	Open Database Connectivity
OLE	object linking and embedding
OLE DB	Object Linking and Embedding Database
PCT	Program Control Table
PDSE	Partitioned Data Set Extended
PIP	program initialization parameter
PLU	primary logical unit
PPT	Program Processing Table
PU	physical unit
PVC	permanent virtual circuit
QLLC	Qualified Logical Link Control
RH	request/response header
RIM	ring indicator monitor

RSP	Response
RTM	Response Time Monitor
RU	request/response unit
RUI	Request Unit Interface
SAP	service access point
SDK	software development kit
SDLC	Synchronous Data Link Control
SID	Security ID
SLI	Session Level Interface
SLU	secondary logical unit
SNA	Systems Network Architecture
SNRM	Set Normal Response Mode
SP	Service Pack
SQL	Structured Query Language
SSCP	System Services Control Point
SVC	switched virtual circuit
TCP/IP	Transmission Control Protocol/Internet Protocol
TCT	Terminal Control Table
TH	transmission header
TN	Telnet
TP	transaction program
TS	transmission service
TSO	time sharing option
VSAM	Virtual Storage Access Method
VTAM	Virtual Telecommunications Access Method
WAN	wide area network
XID	Exchange Identification

Host Integration Server Support

This section explains how Host Integration Server works in Systems Network Architecture (SNA) networks. The following information is included:

- A summary of the types and number of SNA sessions supported by Host Integration Server.
- A list of the SNA commands supported by Host Integration Server. This section also shows the type of SNA sessions in which the commands flow.
- The transmission service (TS) and function management (FM) profiles supported by Host Integration Server.
- A list and explanation of the five categories of SNA sense codes generated by Host Integration Server. This section also describes logical unit (LU) type 6.2-specific sense data.

For specific information about an application programming interface (API) supported by Host Integration Server, see the programmer's guide for that API:

- APPC Programmer's Guide
- CPI-C Programmer's Guide
- LUA Programmer's Guide

In This Section

SNA Session Support

This section lists and explains the types and numbers of SNA sessions supported by Host Integration Server.

In This Section

LU Support

A Host Integration Server node can support as many as 15,000 sessions.

Logical unit 6.2 (LU 6.2) works in combination with node type 2.1 to provide Advanced Program-to-Program Communications (APPC). An LU 6.2 can be either a primary logical unit (PLU) or a secondary logical unit (SLU). The partner LU can reside in a type 2.1 node, within an upstream type 5 node, or within the Host Integration Server node itself.

LU types 0, 1, 2, or 3 are always SLUs, and the partner LU in an upstream type 5 node is always the PLU. LU 0 supports session-level cryptography, within certain limits, on the Request Unit Interface (RUI). You implement session-level cryptography through Cryptography Verification (CRV) requests; the RUI applications must perform all necessary processing. For details about the limits of session-level cryptography and about the processing carried out by the RUI application, see the LUA Programmer's Guide. For all interfaces other than RUI, CRV requests are handled with a negative response.

See Also

Other Resources

SNA Session Support

Session Support

Host Integration Server supports many types of SNA communications sessions:

- A System Services Control Point – physical unit (SSCP-PU)
- A Host Integration Server physical unit (PU) Services Manager which communicates with the System Services Control Point (SSCP) on the host
- An SSCP-LU
- A Host Integration Server logical unit (LU) which communicates with a host SSCP
- A Primary logical unit-secondary logical unit (PLU-SLU)
- A Host Integration Server LU which communicates with host or peer LUs

See Also

Other Resources

Host Integration Server Support

Command Request Support

The next three tables list the SNA command requests supported by Host Integration Server. Note that not all commands are supported for every LU-to-LU session type. For example, Bracket Initiation Stopped (BIS) is only supported for use by an APPC LU.

The following three tables categorize requests in the following order:

1. Session control requests
2. Function management data requests: network services (maintenance) and network services (session)
3. Data flow control requests

The three tables denote the category for each request by the setting of the request/response unit (RU) category bits in the request/response header (RH), and by the function of the request. Host Integration Server does not support commands that fall in the network control request category.

For more information about command requests, see your IBM documentation.

SNA Request Support for Session Control Requests

Session	Request	Direction
SSCP-PU	ACTPU	SSCP to PU
	DACTPU	SSCP to PU
SSCP-LU	ACTLU	SSCP to SLU
	DACTLU	SLU to SSCP
PLU-SLU	BIND	PLU to SLU
	UNBIND	PLU to/from SLU
	CLEAR	PLU to SLU
	SDT	PLU to SLU

SNA Request Support for Function Management Data Requests

Session	Request	Direction
SSCP-PU	NMVT (1)	SSCP to/from PU
	INIT-SELF (2)	SLU to SSCP
	TERM-SELF (2)	SLU to SSCP
	NOTIFY (2)	SLU to SSCP
	NSPE (2)	SSCP to SLU

1. Network services, maintenance services

2. Network services, session services

SNA Request Support for Data Flow Control Requests

Session	Request	Direction

PLU-SLU	BID	PLU to/from SLU
	BIS	PLU to/from SLU
	CANCEL	PLU to/from SLU
	CHASE	PLU to SLU
	LUSTAT	PLU to/from SLU
	RTR	PLU to/from SLU
	SHUTC	PLU from SLU
	SHUTD	PLU to SLU
	SIGNAL	PLU to/from SLU

See Also

Other Resources

Host Integration Server Support

Transmission Service and Function Management Profiles

Host Integration Server enforces the rules that govern active sessions between communicating partners on an SNA network.

The Host Integration Server computer can select some of these session protocols—such as request/response control modes, bracket usage, and pacing—when you activate a session. The term "profiles" refers to specific combinations of these selectable protocol options.

Transmission service (TS) profiles refer to transmission control options, while function management (FM) profiles refer to data flow control and function management data options.

When activated, TS and FM profiles of a session are specified by using parameters in the appropriate session activation request and response such as ACTPU, ACTLU, BIND.

The following table specifies the TS and FM profiles supported by each of the Host Integration Server session types.

Session type	TS profiles	FM profiles
SSCP-LU session	1	0
SSCP-PU session	1	0
3270 display and printer PLU-SLU session	3	3
LU 6.2 PLU-SLU session	7	19
LUA PLU-SLU session	2, 3, 4	2, 3, 4, 7, 18

In This Section

TS Profile 1

Transmission service (TS) Profile 1 is supported on System Services Control Point physical unit (SSCP-PU) and System Services Control Point secondary logical unit (SSCP-SLU) sessions; the profile does not require a TS usage field. This profile specifies the following session rules:

- Pacing is not supported.
- Identifiers, rather than sequence numbers, are used on the normal flows.
- ACTPU, DACTPU, ACTLU, and DACTLU are supported.
- Maximum outbound request/response (RU) size is 256 bytes.
- Maximum inbound RU size is 512 bytes.

See Also
Other Resources
Transmission Service and Function Management Profiles

TS Profile 2

Transmission service (TS) Profile 2 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU 0. This profile specifies the following session rules:

- Primary logical unit-secondary logical unit (PLU-SLU) and SLU-PLU normal flows are paced.
- Sequence numbers are used on the normal flows whenever the transmission header (TH) format used includes a sequence number field.
- CLEAR is supported.
- SDT, RQR, STSN, and CRV are not supported.

The TS usage subfields defining the options for this profile are the following:

- Pacing window counts
- Maximum request/response (RU) sizes on the normal flows

See Also

Other Resources

Transmission Service and Function Management Profiles

TS Profile 3

Transmission service (TS) Profile 3 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU types 0, 1, 2, or 3. Selection is a BIND option. This profile specifies the following session rules:

- PLU-SLU and SLU-PLU normal flows are paced.
- Sequence numbers are used on the normal flows.
- BIND, UNBIND, CLEAR, and SDT are supported.

The TS usage subfields defining the options for this profile are the following:

- Pacing counts
- Maximum request/response (RU) sizes on the normal flows

See Also

Other Resources

Transmission Service and Function Management Profiles

TS Profile 4

Transmission service (TS) Profile 4 is supported on logical unit-logical unit (LU-LU) sessions using LU types 0 or 1. This profile specifies the following session rules:

- Primary logical unit-secondary logical unit (PLU-SLU) and SLU-PLU normal flows are paced.
- Sequence numbers are used on the normal flows whenever the transmission header (TH) format used includes a sequence number field.
- SDT, CLEAR, RQR, and STSN are supported.
- CRV is supported when session-level cryptography is selected via a BIND parameter.

The TS usage subfields defining the options for this profile are the following:

- Pacing window counts
- Maximum request/response (RU) sizes on the normal flows

See Also

Other Resources

Transmission Service and Function Management Profiles

TS Profile 7

Transmission service (TS) Profile 7 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU 6.2. This profile specifies the following session rules:

- PLU-SLU and SLU-PLU normal flows are paced.
- Sequence numbers are used on the normal flows.
- BIND and UNBIND are supported.

The TS Usage subfields defining the options for this profile are the following:

- Pacing counts
- Maximum request/response (RU) sizes on the normal flows

See Also
Other Resources
Transmission Service and Function Management Profiles

FM Profile 0

Function management (FM) Profile 0 is supported on SSCP-PU and SSCP-SLU sessions. This profile specifies the following session rules:

- SSCP, PU, and SLU use immediate request mode and immediate response mode.
- Only single-RU chains are allowed.
- All chains require definite response.
- No compression.
- No data flow control (DFC) RUs.
- No function management (FM) headers.
- No brackets.
- No alternate code.
- Normal flow send/receive mode is half-duplex contention.
- Secondary wins contention.
- SSCP is responsible for recovery.

See Also

Other Resources

Transmission Service and Function Management Profiles

FM Profile 2

Function management (FM) Profile 2 is supported on primary logic unit-secondary logic unit (PLU-SLU) sessions using LU 0. This profile uses the following session rules:

- SLU uses delayed request mode.
- SLU uses immediate response mode.
- Only single-RU chains are allowed.
- SLU requests indicate no-response.
- No FMH-1 SCB compression.
- Length-checked compression allowed.
- No data flow control (DFC) RUs.
- No FM headers.
- SLU is the first speaker if brackets are used.
- Bracket termination rule 2 is used if brackets are used.
- PLU will send an end bracket (EB).
- SLU will not send an EB.
- Normal-flow send/receive mode is full duplex (FDX).
- PLU is responsible for recovery.

The following FM Usage fields define the options for Profile 2:

- Primary request control mode selection
- Primary chain response protocol (no-response cannot be used)
- Brackets usage and reset state
- Alternate code

See Also

Other Resources

Transmission Service and Function Management Profiles

FM Profile 3

Function management (FM) Profile 3 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU types 0, 1, 2, or 3. This profile specifies the following session rules:

- PLU and SLU use immediate response mode.
- PLU and SLU support the following data flow control (DFC) commands (an asterisk [*] indicates commands permitted by the FM profile that are never sent by Host Integration Server):
    - **CANCEL**
    - **SIGNAL**
    - **LUSTAT** (SLU-PLU only)
    - **CHASE***
    - **SHUTD**
    - **SHUTC**
    - **RSHUTD***
    - **BID** and **RTR**

The following FM Usage fields define the options for Profile 3:

- Chaining use (PLU and SLU)
- Request control mode selection (PLU and SLU)
- Chain response protocol (PLU and SLU)
- Compression indicator (PLU and SLU)
- Send EB indicator (PLU and SLU)
- FM header usage
- Bracket usage
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode

The next three tables list the FM Profile 3 options that Host Integration Server supports. Host Integration Server rejects any BIND option that specifies an option value not listed in these tables.

Supported options for BIND requests from primary LUs

Usage option	Value	Meaning
Chaining use element	0	Can send only single-element chains

	1	Can send single- or multiple-chains
Request control mode	0	Immediate request mode used
Chain response protocol	01	Can request only exception-only responses
	10	Can request only definite responses
	11	Can request definite or exception-only responses
Compression indicator	0	Cannot send compressed data
	1	Can send compressed data (LU 1 only)
Send EB indicator	1	Can send an end bracket

Supported options for BIND requests from secondary LUs

Usage option	Value	Meaning
Chaining use element	0	Can only send single-element chains (not LU 2)
	1	Can send single- or multiple-chains
Request control mode	0	Immediate request mode used
Chain response protocol	01	Can request only exception-only responses
	10	Can request only definite responses
	11	Can request definite or exception-only responses
Compression indicator	0	Cannot send compressed data
Send EB indicator	1	Cannot send end bracket

Supported common protocol values for BIND requests from any LU

Usage option	Value	Meaning
FM header usage	0	Cannot exchange FM headers
	1	Can exchange FM headers
Bracket usage	1	Bracket protocols must be used
Bracket termination rule	1	Bracket termination rule 1 is used
Alternate code set allowed	0	Must use Extended Binary Coded Decimal Interchange Code (EBCDIC)
Normal flow send/receive mode	10	Half-duplex flip-flop

See Also

Other Resources

Transmission Service and Function Management Profiles

FM Profile 4

Function management (FM) Profile 4 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU types 0 or 1. This profile uses the following session rules:

- PLU and SLU use immediate response mode.
- PLU and SLU support the following data flow control (DFC) commands (an asterisk [*] indicates commands permitted by the FM profile that are never sent by Host Integration Server):
    - **CANCEL**
    - **SIGNAL**
    - **LUSTAT**
    - **QEC**
    - **QC**
    - **RELQ**
    - **CHASE***
    - **SHUTD**
    - **SHUTC**
    - **RSHUTD***
    - **BID** and **RTR** (allowed only if brackets are used)
- Length-checked compression allowed

The following FM Usage fields define the options for Profile 4:

- Chaining use (PLU and SLU)
- Request control mode selection (PLU and SLU)
- Chain response protocol (PLU and SLU)
- FMH-1 SCB (String Control Byte) Compression indicator (PLU and SLU)
- Send EB indicator (PLU and SLU)
- FM header usage
- Brackets usage and reset state
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode

- Recovery responsibility
- Contention winner/loser
- Half-duplex flip-flop reset states

See Also

Other Resources

Transmission Service and Function Management Profiles

FM Profile 7

Function management (FM) Profile 7 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU 0. This profile uses the following session rules:

- PLU and SLU use immediate response mode.
- PLU and SLU support the following data flow control (DFC) commands (an asterisk [*] indicates commands permitted by the FM profile that are never sent by Host Integration Server):
    - **CANCEL**
    - **SIGNAL**
    - **LUSTAT**
    - **RSHUTD***
- Length-checked compression is allowed on LU 0 only.

The following FM Usage fields define the options for Profile 7:

- Chaining use (PLU and SLU)
- Request control mode selection (PLU and SLU)
- Chain response protocol (PLU and SLU)
- FMH-1 SCB Compression indicator (PLU and SLU)
- Send EB indicator (PLU and SLU)
- FM header usage
- Brackets usage and reset state
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode
- Recovery responsibility
- Contention winner/loser
- Half-duplex flip-flop reset states

See Also
Other Resources
Transmission Service and Function Management Profiles

FM Profile 18

Function management (FM) Profile 18 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU 0. This profile uses the following session rules:

- PLU and SLU use immediate response mode.
- PLU and SLU support the following data flow control (DFC) commands (an asterisk [*] indicates commands permitted by the FM profile that are never sent by Host Integration Server):
    - **CANCEL**
    - **SIGNAL**
    - **LUSTAT**
    - **BIS** and **SBI** (allowed only if brackets are used)
    - **CHASE***
    - **BID** and **RTR** (allowed only if brackets are used)
- Length-checked compression is allowed.

The following FM Usage fields define the options for Profile 18:

- Chaining use (PLU and SLU)
- Request control mode selection (PLU and SLU)
- Chain response protocol (PLU and SLU)
- FMH-1 SCB Compression indicator (PLU and SLU)
- Send EB indicator (PLU and SLU)
- FM header usage
- Brackets usage and reset state
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode
- Recovery responsibility
- Contention winner/loser
- Half-duplex flip-flop reset states

See Also
Other Resources

FM Profile 19

Function management (FM) Profile 19 is supported on primary logical unit-secondary logical unit (PLU-SLU) sessions using LU 6.2. This profile specifies the following session rules:

- PLU and SLU use immediate request mode.
- PLU and SLU use immediate response mode.
- Multiple RU chains are allowed.
- PLU and SLU requests indicate definite or exception response.
- No compression is used.
- Brackets are used.
- FM headers (types 5, 7, and 12 only) are allowed.
- Conditional termination for brackets, specified by conditional end bracket (CEB),is used.
- PLU and SLU may send CEB.
- Normal-flow send/receive mode is half-duplex flip-flop.
- Half-duplex flip-flop reset state is SEND for PLU.
- Symmetric error responsibility.
- Contention winner/loser is negotiated at BIND time.
- PLU and SLU support the following data flow control (DFC) commands:
    - **SIGNAL**
    - **LUSTAT**
    - **BIS**
    - **RTR**
- The following combinations of RQE, RQD, CEB, and CD are allowed on end-chain Russ

RQE	CD	CEB
RQD2	CD	CEB
RQD3	CD	CEB
RQE1	CD	CEB
RQD	CD	CEB

The only option for Profile 19 is contention winner/loser.

See Also

Other Resources

Transmission Service and Function Management Profiles

Option Set Support

The following table lists the option sets supported by Host Integration Server. The reference numbers are those specified in IBM documentation.

Ref No.	Option Set
101	Flush the send buffer of the LU
102	Get attributes
103	Post on receipt with test for posting (supported by asynchronous APPC verbs)
104	Post on receipt with wait (supported by asynchronous APPC verbs)
105	Prepare to receive
106	Receive immediate
109	Get TP name and instance identifier
110	Get conversation type
201	Queued allocation of a contention winner session
203	Immediate allocation of a session
204	Conversations between programs located at the same LU
205	Queued allocation for when session is free
211	Session-level LU-LU verification
212	User ID verification
213	Program-supplied user ID and password
214	User ID authorization
241	Send program initialization parameter (PIP) data
243	Accounting
244	Long locks
245	Test for request-to-send received
246	Data mapping
251	Extract transaction and conversation identification information
290	Logging of data in a system log
291	Mapped conversation LU services component

505	LU definition verbs (operator only)
601	MIN_CONWINNERS_TARGET parameter
602	RESPONSIBLE(TARGET) parameter
603	DRAIN_TARGET(NO) parameter
605	LU-LU session limit
606	Locally known LU names
607	Uninterpreted LU names
610	Maximum RU size bounds
612	Automatic activation limit for contention winner

See Also

Other Resources

Host Integration Server Support

SNA Sense Codes

If there is a communication problem during an SNA session, Host Integration Server can generate sense codes that notify the remote system of the type of problem. The sense codes fall into five distinct categories, corresponding to the type of problem that occurred.

Value	Category
X'08'	Request Reject
X'10'	Request Error
X'20'	State Error
X'40'	RH Usage Error
X'80'	Path Error

The following sections list the sense codes by category, and include two additional categories of sense codes:

- Sense codes sent by a 3270 emulator on LUSTAT requests
- Sense codes that can only flow on LU 6.2 sessions

In This Section

Request Reject (Category X'08')

This category indicates that the request was delivered to the intended half session, and was understood and supported, but not executed. The following codes are in this category.

Code	Meaning
0801	Resource not available
0802	Intervention required (LU 1/LU 3 printer soft error)
0805	Session limit exceeded
0809	Mode inconsistency
080A	Permission rejected, NOTIFY will not be sent
080B	Bracket race error
080C	Procedure not supported
0812	Insufficient resources
0814	Bracket bid reject-RTR forthcoming
0815	Function active
081B	Contention race condition
081C	Request not executable (hard error)
0821	Invalid session parameters
0829	Change direction required
082B	Presentation space integrity lost
082D	LU busy (usually local copy print in progress)
082E	Intervention required (local copy print soft error)
082F	Request not executable (local copy print hard error)
0831	LU component disconnected
0843	WCC print command not sent (RQD or RQE, CD, EB)
0845	Permission rejected, NOTIFY will be sent
0863	Referenced character set does not exist
0871	Read partition state error (SLU in retry state)

See Also
Other Resources
SNA Sense Codes

Request Errors (Category X'10')

Sense codes in this category represent an imbalance in half-session capabilities. They indicate that the RU (request/response unit) was delivered to the intended half-session but could not be processed. The following codes are in this category.

Code	Meaning
1003	Function not supported (also used instead of X'0826', which is not supported by Host Integration Server)
1005	Parameter modifying a control function is invalid
1007	Category not supported (SSCP data for host printer)

See Also
Other Resources
SNA Sense Codes

State Errors (Category X'20')

Sense codes in this category indicate a sequence number error, or a request/response header (RH) or request/response unit (RU) that is not allowed in the receiver's current state. This condition prevents delivery of the request to the intended half-session. The following codes are in this category.

Code	Meaning
2001	Sequence number error
2002	Chaining error
2003	Bracket error
2004	Direction error
2005	Data traffic reset
2006	Data traffic quiescent
2007	Data traffic not quiescent

See Also
Other Resources
SNA Sense Codes

RH Usage Errors (Category X'40')

These sense codes indicate that fields in the request/response header (RH) are contrary to SNA rules or previously selected BIND options and prevent the intended half-session from receiving the request.

The checks 400A and 4012 are also performed. These do not permit negative responses because they apply to no-response sessions and responses, respectively.

The following codes are in this category.

Code	Meaning
4006	Exception response not allowed
4007	Definite response not allowed
400F	Incorrect use of format indicator
4011	Incorrect specification of RU category
4014	Incorrect use of DR1, DR2, ER

See Also

Other Resources

SNA Sense Codes

Path Errors (Category X'80')

Sense codes in this category indicate that the request could not be delivered to the required half-session because of path errors. The following codes are in this category.

Code	Meaning
8004	Unrecognized destination field address (DAF)
8005	No session
8006	Invalid format identification (FID): Error detected and logged; negative response not sent
8007	Segmentation error request/response header (RH) not present or too short: negative response sent only for segmentation error
8008	Primary unit (PU) not active: SSCP-PU was not active and request was not ACTPU or DACTPU
8009	Logical unit (LU) not active: DAF specified an LU for which the SSCP-SLU session has not been activated, and request was not ACTLU or DACTLU
800F	Invalid address combination

See Also

Other Resources

SNA Sense Codes

LUSTAT Sense Codes

The following sense codes are sent by a 3270 emulator on LUSTAT requests.

Code	Meaning
0001 B000	Host initialized local copy soft error recovery
081C 0000	Soft error changed to hard error (LU 1/LU 3)
081C B000	Soft error changed to hard error (local copy)
082B 0000	SLU session changed from SSCP to PLU (SYSREQ)

See Also

Other Resources

SNA Sense Codes

Sense Data Specific to LU 6.2

The LU 6.2 sense codes are specified and interpreted by SNA components within Host Integration Server, and, in general, are presented as return codes and as parameters to specific verbs.

The following sense codes are carried on responses flowing on LU 6.2 sessions.

Code	Meaning
0814	Bracket bid reject RTR forthcoming
0819	RTR not required receiver of RTR request has nothing to send
0835	Invalid parameter request contained a fixed or variable field that is invalid. Bytes 2 and 3 contain a 2-byte binary count which indexes the field error (zero origin)
0846	ERP message forthcoming: The received request was rejected for a reason to be specified in a forthcoming FMH-7 request which itself carries SNA sense data (see the following table for a list of these codes)
088B	BB not accepted BIS reply requested
2010	BIS protocol error receiver detected a BIS error, for example, two BIS replies received

The following data is carried on an FMH-7 request, coming after the flow of a negative response X'0846'.

Code	Meaning
1008 600B	Resource failure no retry
1008 6021	Allocation error TPN not recognized
1008 6031	Allocation error PIP not allowed
1008 6032	Allocation error PIP not specified correctly
1008 6034	Allocation error conversation type mismatch
1008 6041	Allocation error sync level not supported by pgm
080F 6051	Allocation error security not valid
084B 6031	Allocation error trans pgm not available retry
084C 0000	Allocation error trans pgm not available no retry
0864 0000	Deallocate abend prog
0864 0001	Deallocate abend svc
0846 0002	Deallocate abend timer

0889 0000	Prog error no trunc or prog error purging
0889 0001	Prog error trunc
0889 0100	Svc error no trunc or svc error purging
0889 1001	Svc error trunc

See Also

Other Resources

SNA Sense Codes

Network Management

This section describes the support that Host Integration Server provides for centralized network management.

In This Section

Connections Used for NetView and RTM

Link Alerts for SDLC and Token Ring

Link Statistics

Alerts Used by Applications, NVAlert, and NVRunCmd

Local Logging of Network Management Data

For information about how to configure Host Integration Server to work with NetView, and information about configuring the NVAlert and NVRunCmd services, see Network Integration User's Guide.

Connections Used for NetView and RTM

Host Integration Server can make use of NetView, a reporting system that runs on an IBM host (mainframe). NetView sends alerts between the host and the PCs that connect to it. Host Integration Server can also receive information from Response Time Monitor (RTM), a 3270 and NetView facility that monitors the amount of time it takes for a host to respond during 3270 display sessions.

To send link alerts, link statistics, and application-generated alerts to the NetView program at the host, you must specify the connection on which the alerts will be sent. See the Host Integration Server online Help for information about specifying the connection.

RTM data and NetView alerts sent to the host by 3270 users are not necessarily sent via the NetView connection. RTM data is sent on the connection for the session to which it refers, while 3270 user alerts are sent on the connection for the currently selected 3270 session.

For information about configuring SNA Management to work with NetView and information about configuring the NVAlert and NVRunCmd services, see Network Integration User's Guide.

See Also

Other Resources

Network Management

Link Alerts for SDLC and Token Ring

When a Synchronous Data Link Control (SDLC) or Token Ring connection fails, Host Integration Server logs diagnostic information, called link alerts, about the connection failure. The alerts are logged in a log file that can be viewed using the Windows Event Log service. To find a Host Integration Server log, in the Event Log service, on the Log menu, make sure Application is selected. Under the "Source" heading, the application log file will have a name starting with "SNA."

The link alerts are also used to build a network management vector transport (NMVT) alert, which includes probable causes and suggested actions for the alert. If a connection on the server running Host Integration Server has been designated for NetView, and the connection is active, the NMVT alert will be sent on that connection.

This section includes information that includes an explanation of the general format of link alerts, common to both SDLC and Token Ring, descriptions of individual alerts produced by the SDLC link service, detailed information about the local logging of SDLC alerts, descriptions of individual alerts produced by the Token Ring link service, and details of the local logging of Token Ring alerts.

In This Section

For information about the NetView vectors supported by the NVAlert and NVRunCmd services, see Alerts Used by Applications, NVAlert, and NVRunCmd.

Link Alert Format and Common Subvectors

The general format of link alerts is as follows:

```
NMVT HEADER
41 03 8D 00 00 00 00 00     NMVT Header
Major Vector Header
LL LL     Length of Major Vector
00 00     Alert major vector
```

The specific alert is identified by the alert description and alert ID (part of the generic alert subvector). Common subvectors are listed later in this section. SDLC and Token Ring failure alerts are described in the SDLC Failure Alerts and Token Ring Failure Alerts.

Each subvector or subfield begins with a 1-byte length field and a 1-byte key identifying the subvector or subfield. The length field contains the length in bytes of the entire subvector or subfield, including the length and key bytes. Where the length byte is shown as LL (record length field), this indicates that the length is not fixed because different alerts require different data to be included.

The following table lists all the possible subvectors used in the alert, with their subvector keys (identifiers).

Link alert subvectors

Subvector key (identifier)	Subvector
05	Hierarchy/Resource List Subvector
03	Hierarchy Name List Subvector
01	Date/Time Subvector
10	Product Set ID Subvector (this appears twice, as Alert Sender PSID and Indicated Resource PSID).
51	LAN LCS Data Subvector
52	LCS Configuration Data Subvector
8C	Link Station Data Subvector
91	Basic Alert Subvector
92	Generic Alert Subvector
93	Probable Cause Subvector
94	User Causes Subvector
95	Install Causes Subvector
96	Failure Causes Subvector
A1	Detail Qualifier Subvector

The following tables describe the formats of link alert subvectors. Additional tables, in this section describe the individual link alerts.

Hierarchy/resource list subvector

Format and contents	Description
LL 05	Hierarchy/Resource List Subvector
LL 10	Hierarchy/Resource List Subfield
hh	Flags: 80 (1-User) 00 (Server)
LL	Name field length: name length plus this length byte.
hh...hh	Adapter name in EBCDIC, 8 bytes
00	Flags byte: always 0
3F	Resource type: port
Optional second name	Remote host name. For Token Ring, use the local ring name. See the individual alert descriptions found later in this section.
LL	Name field length: name length plus this length byte.
hh...hh	Name in EBCDIC, up to 8 bytes
00	Flags byte: always 0
F4 or 2E	Resource type: control point or Token Ring.

Hierarchy/name list subvector

Format and contents	Description
LL 03	Hierarchy Name List subvector
00 or 03	Reserved byte
hh	Number of names in the hierarchy name list.
LL	Length of each name, including this length byte.
hh...hh	Name of resource for each name.
D3 C3 D6 D5	Resource type identifier: LCON (logical link connection not known to SNA)

Date/time subvector alerts

Format and contents	Description
0A 01	Date/Time Subvector
08 10	Local Date/Time subfield
hh	Year two digits
hh	Month
hh	Date
hh	Hours

hh	Minutes
hh	Seconds (binary)

Product set ID subvector

Format and contents	Description
2A 10	Product Set ID subvector
00	(unused field)
27 11	Product ID subvector
0C	Product classification: non-IBM software
08 04	Software product common level subfield
F0 F2	Version identifier: 02
F0 F0	Release identifier: 00
F0 F0	Modification identifier: 00
13 06	Software product common name subfield
C4 C3 C1 61 D4 E2 40 C3 D6 D4 D4 40 E2 C5 D9 E5 D9	Product name: Microsoft Host Integration Server
09 08	Software product program number subfield
F0 F0 F0 F0 F0 F0 F0	Product program number: zeros

LAN link connection subsystem data subvector*

Format and contents	Description
LL 51	LAN LCS Data Subvector
04 02	Ring Identifier subfield
hh hh	Ring number
08 03	Local Individual MAC Address subfield
hh...hh	Local individual MAC address
08 04	Remote Individual MAC Address subfield
hh...hh	Remote individual MAC address
LL 05	LAN Routing Information subfield
hh...hh	LAN routing information

* Token ring only.

Link connection subsystem configuration data subvector

Format and contents	Description

LL 52	LCS Configuration Data Subvector
04 01	Port Address subfield
hh hh	Port address
03 02	Remote Device Address subfield
hh	Remote device address
03 04	Local Device Address subfield
hh	Local device address
04 06	Link Station Attributes subfield
01 or 02 or 03	Link station role: Primary, Secondary, or Negotiable
03 or 04	Remote node type: Type 4 or Type 2.1
06 07	Link Attributes subfield
01 or 02	Link connection type: nonswitched or switched
01 or 02	Half-duplex or full-duplex
01	Protocol: SDLC
01 or 02	Point-to-point or multipoint

Link station data subvector

Format and contents	Description
LL 8C	Link Station Data Subvector
04 01	N(s) and N(r) counts subfield
hh	N(s) count
hh	N(r) count
03 02	Outstanding frame count subfield
hh	Outstanding frame count
04 03	Last SDLC Control Field Received subfield
hh hh	Last control field received
04 04	Last SDLC Control Field Sent subfield
hh hh	Last control field sent

03 05	Sequence Number Modulus subfield
80	Sequence number modulus
03 06	Link Station State subfield
80 or 40	Link station state: local or remote station sending RNR
04 07	LLC Reply Timer Expiration Count subfield
hh hh	Reply timer expiration count
03 08	Last Received N(r) Count subfield
hh	Last received N(r) (binary)

Basic alert subvector

Format and contents	Description
0E 91	Basic Alert subvector
00	Not-operator initiated
01	Alert type: permanent
0E	General cause code: link-level protocol error
00 80 or 00 34	Specific component code: Token Ring LAN or SDLC data link control
00 00	Alert description code
00 00	User action code
00 00	Detail text reference code
00	Unused field

Generic alert subvector

Format and contents	Description
0B 92	Generic Alert subvector
00 00	Flags: not operator-initiated, not a held alert
01	Alert type: permanent
hh hh	Alert description: See alert descriptions.
hh hh hh hh	Alert ID number (unique to a particular alert)

Probable cause subvector

Format and contents	Description
LL 93	Probable cause subvector
hh hh	Probable cause: See individual alert descriptions. This alert may include more than one entry.

User causes subvector

Format and contents	Description
LL 94	User Causes Subvector
LL 01	User causes subfield
hh hh	User cause: See individual alert descriptions. More than one entry may appear here.
LL 81	Recommended actions subfield
hh hh	Recommended action: See individual alert descriptions. More than one entry may appear here.
03 83	Product set ID index subfield
99	Product Set ID index
LL 82	Detailed data subfield. This subfield may appear more than once.
99	Product Set ID code
hh	Data ID: See individual alert descriptions.
00	Data encoding: hexadecimal
hh...hh	Detailed data: See individual alert descriptions.

Install cause subvector

Format and contents	Description
LL 95	Install Causes Subvector
LL 01	Install causes subfield
hh hh	Install cause: See individual alert descriptions. More than one entry may appear here.
LL 81	Recommended actions subfield
hh hh	Recommended action: See individual alert descriptions. More than one entry may appear here.
03 83	Product set ID index subfield
99	Product Set ID index
LL 82	Detailed data subfield. This subfield may occur more than once.
99	Product Set ID code
Hh	Data ID: See individual alert descriptions.
00	Data encoding: hexadecimal
Hh...hh	Detailed data: See individual alert descriptions.

Failure cause subvector

Format and contents	Description

LL 96	Failure Causes Subvector
LL 01	Failure causes subfield
hh hh	Failure cause: See individual alert descriptions. More than one entry may appear here.
LL 81	Recommended actions subfield
hh hh	Recommended action: See individual alert descriptions. More than one entry may appear here.
03 83	Product set ID index subfield
99	Product Set ID index
LL 82	Detailed data subfield. This subfield may occur more than once.
99	Product Set ID code
hh	Data ID: See individual alert descriptions.
00	Data encoding: hexadecimal
hh...hh	Detailed data: See individual alert descriptions.

Detail qualifier (hexadecimal) subvector

Format and contents	Description
06 A1	Detail Qualifier subvector
hh hh hh hh	Detail Qualifier subfield four bytes as follows: Token ring: adapter number Alert number Two-byte error code SDLC: link ID Station address, outage

SDLC Failure Alerts

The tables in this section describe the alerts generated by the SDLC link service provided with Host Integration Server. The alerts are listed in numerical order by alert description and within each description in numerical order by alert ID.

Write timeout alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	1000	Equipment malfunction
Alert ID:	61D0 2E7B	
Probable causes:	2031	Line
	3601	Local modem
	2200	Remote node
	3603	Remote modem
Failure causes:	3511	Line
	3601	Local modem
	2200	Remote node
	3603	Remote modem
	F034	CTS failed to rise
Recommended actions:	0403	Run modem tests
	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	2E	Write timeout retry exceeded

Abnormal modem response alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	1000	Equipment malfunction
Alert ID:	70A1 5CB0	
Probable causes:	3601	Local modem
	3401	Local cable
	3302	Communications adapter
User causes:	3401	Cabling installed incorrectly
Recommended actions:	0301	Check cable and its connections

Failure causes:	3601	Local modem
	3401	Local cable
	3302	Communications adapter
	F035	DSR dropped
Recommended actions:	0403	Run modem tests
	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	2D	Abnormal response

DSR failure alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	1000	Equipment malfunction
Alert ID:	83BB 6EC8	
Probable causes:	3601	Local modem
	3401	Local cable
	3302	Communications adapter
User causes:	3400	Cable loose
	0205	Local modem power off
Failure causes:	3601	Local modem
	3401	Local cable
	3302	Communications adapter
	F035	DSR dropped
Recommended actions:	0403	Run modem tests
	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	11	DSR failure

DCD failure alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	1000	Equipment malfunction
Alert ID:	C3BB 504A	

Probable causes:	2031	Line
	3601	Local modem
Failure causes:	3511	Line
	3601	Local modem
	F038	Carrier detect lost
Recommended actions:	0403	Run modem tests
	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	14	DCD failure

Invalid frame received: i-field not permitted alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	1103 D152	
Probable causes:	2004	SDLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F021	I-field received when not permitted
Recommended actions:	2011	Review hexadecimal display of alert report
	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	87	Invalid frame received

Invalid frame received: invalid/unsupported cmd/rsp alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	15C2 CCE5	
Probable causes:	2004	SDLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F020	Invalid/unsupported command or response received.
Recommended actions:	2011	Review hexadecimal display of alert

	32A0	Report the following detail qualifier: Outage qualifier
Detail qualifier:	87	Invalid frame received

Invalid frame received: invalid n(r) received alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	1C40F78B	
Probable causes:	2004	SDLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F022	Invalid N(R) received
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	87	Invalid frame received

RIM/RD received alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	29BF 0032	
Probable causes:	2104	SDLC communications/remote node
	1023	Communications program in remote node
Failure causes:	2104	SDLC communications/remote node
	3511	Line
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	88	RIM received
	89	RD received

Invalid frame received: max i-field length exceeded alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	5F76 24FD	
Probable causes:	2104	SDLC communications/remote node

	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F023	Received I-field exceeded maximum length
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	87	Invalid frame received

FRMR received: no reason given alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	A472 BC48	
Probable causes:	2004	SDLC communications
	1000	Software program
Failure causes:	1000	Software program
	F014	FRMR received: no reason specified
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	86	FRMR received

FRMR received: i-field received when not permitted

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	B3B7 D723	
Probable causes:	2004	SDLC communications
	1000	Software program
Failure causes:	1000	Software program
	F011	FRMR received: I-field sent when not permitted
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	86	FRMR received

FRMR received: invalid/unsupported command or response alert

Name of subvector	Hexadecimal value in subvector	Description

Alert description:	2100	Software program error
Alert ID:	B776 CA94	
Probable causes:	2004	SDLC communications
	1000	Software program
Failure causes:	1000	Software program
	F010	FRMR received: invalid/unsupported command or response sent
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	86	FRMR received

FRMR received: max i-field length exceeded alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	BA35 EC4D	
Probable causes:	2004	SDLC communications
	1000	Software program
Failure causes:	1000	Software program
	F013	FRMR received: maximum I-field length exceeded
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	86	FRMR received

FRMR received: invalid n(r) sent alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	BEF4 F1FA	
Probable causes:	2004	SDLC communications
	1000	Software program
Failure causes:	1000	Software program
	F012	FRMR received: invalid N(R) sent
Recommended actions:	2011	Review hexadecimal display of alert

	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	86	FRMR received

DISK/SNRM received: snrm received in nrm alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	D635 CA1E	
Probable causes:	2004	SDLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F015	SNRM received while in NRM
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	15	Connection terminated by host

CTS failure alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	0231 CF0A	
Probable causes:	2031	Line
	3601	Local modem
Failure causes:	3511	Line
	3601	Local modem
	F030	CTS dropped
Recommended actions:	0403	Run modem tests
	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	12	CTS failure

Connection problem: xid (exchange identification) retransmission alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	0AEC C2A6	

Probable causes:	2104	SDLC communications/remote node
	2031	Line
User causes:	0209	Remote device power off
Recommended action:	0200	Check power
Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F018	XID poll count exhausted

Idle timeout alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	0E2D DF11	
Probable causes:	2104	SDLC communications/remote node
	2031	Line
User causes:	0209	Remote device power off
Recommended action:	0200	Check power
Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F019	Inactivity timer expired
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	25	Idle timeout retry exceeded

Poll count exhausted alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	32A3 7F1B	
Probable causes:	2104	SDLC communications/remote node
	2031	Line
User causes:	0209	Remote device power off
Recommended action:	0200	Check power

Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F017	Poll count exhausted
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	81	Disconnect retry limit
	82	Contact retry limit
	83	Poll retry limit
	84	No response retry limit

Remote busy retry alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	66D6 DA74	
Probable causes:	2104	SDLC communications/remote node
User causes:	0209	Remote device power off
Recommended actions:	0200	Check power
Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F00F	RNR received threshold reached
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	85	Remote busy retry limit

DM Received in Information Transfer Alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	BD84 C4C9	
Probable causes:	2004	SDLC communications
Failure causes:	2004	SDLC communications
	F01A	DM received

Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	80	DM received

Nonproductive receive timeout alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	C458 E284	
Probable causes:	2104	SDLC communications/remote node
	2031	Line
User causes:	0209	Remote device power off
Recommended action:	0200	Check power
Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F00E	NPR timeout retry limit reached
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	24	Nonproductive receive retry exceeded

DISK/SNRM received: disk received alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	D9C4 172B	
Probable causes:	2004	SDLC communications
Failure causes:	2004	SDLC communications
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	15	Connection terminated by host

Connection problem: i-frame retransmission alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3300	Link error
Alert ID:	E3DA E42F	

Probable causes:	2104	SDLC communications/remote node
	2031	Line
User causes:	0209	Remote device power off
Recommended action:	0200	Check power
Failure causes:	2104	SDLC communications/remote node
	3511	Line
	F06A	Transmit retry limit reached
Recommended actions:	2011	Review hexadecimal display of alert
	32A0	Report the following detail qualifier: Outage qualifier.
Detail qualifier:	29	Connection problem

See Also

Other Resources

Link Alerts for SDLC and Token Ring

SDLC Alert Local Logging

Before Host Integration Server attempts to build and send an SDLC link alert, information about the outage that caused the alert is recorded in a log file that can be viewed using the Windows Event Log service. Message 182 is always logged, as follows:

```
182I:  Connection failure, code = (outagecode)
Cause Data                          = hh...hh
Link Role                           = hh
Remote Node Type                    = hh
```

The Outage Code (outagecode) is also reported on Message 23. For a complete list of these codes, the conditions they represent, and the specific link alerts on which they are used, see the next section.

Cause Data is a list of the failure causes from the Failure causes subvector in the alert. Link Role and Remote Node type are as described in the Link Connection Subsystem Configuration Data subvector in the alert.

A connection failure can be due to a station outage, rather than a link outage. If it is due to a station outage which can be recognized by outage code values of X'80' or higher then Message 183, formatted as follows, is also logged:

```
183I:
Detailed diagnostic data for station (stationaddress):
SBSY   CFTX   CFRX   V(S)   V(R)   N(R)   OSFC   T1CT
hh   hhhh   hhhh   hh     hh     hh     hh     hhhh
```

The value (stationaddress) is the station address as given in the Detail qualifier subvector in the alert.

The other fields are all data from the Link Station Data subvector, as follows:

Field	Description
SBSY	Link station state: 80 or 40
	(local or remote station sending RNR)
CFTX	Last SDLC control field sent
CRX	Last SDLC control field received
V(S)	N(S) count
V(R)	N(R) count
N(R)	Last received N(R)
OSFC	Outstanding frame count
T1CT	Reply timer expiration count

See Also

Other Resources

Link Alerts for SDLC and Token Ring

Identifying Alerts from Local Logs Only

When only the local log of an alert is available, you can still obtain the information that would have been supplied on the alert. To do this, check the outage code given on Message 182 against the following list. Find the alert that uses this outage code, and then check the alert data in SDLC Failure Alerts. Some Outage Codes are used for more than one alert. In these cases, you can find the appropriate alert by comparing the Failure causes codes given on Message 182 with those given in the alert data.

SDLC outage codes

Outage code	Condition	Alert description/ID
11	DSR failure	1000 : 83BB6EC8
12	CTS failure	3300 : 0231CF0A
14	DCD failure	1000 : C3BB504A
15	Connection terminated by host	2100 : D635CA1E 2100 : D9C4172B
24	Nonproductive receive retry exceeded	3300 : C458E284
25	Idle timeout retry exceeded	3300 : 0E2DDF11
29	Connection problem	3300 : 0AECC2A6 3300 : E3DAE42F
2D	Abnormal response	1000 : 70A15CB0
2E	Write timeout retry exceeded	1000 : 61D02E7B
80	DM received	3300 : BD84C4C9
81	Disconnect retry limit	3300 : 32A37F1B
82	Contact retry limit	3300 : 32A37F1B
83	Poll retry limit	3300 : 32A37F1B
84	No response retry limit	3300 : 32A37F1B
85	Remote busy retry limit	3300 : 66D6DA74
86	FRMR received	2100 : A472BC48
		2100 : B3B7D723
		2100 : B776CA94
		2100 : BA35EC4D
		2100 : BEF4F1FA
87	Invalid frame received	2100 : 1103D152
		2100 : 15C2CCE5
		2100 : 1C40F78B
		2100 : 5F7624FD
88, 89	RIM or RD received	2100 : 29BF0032

See Also

Other Resources

Link Alerts for SDLC and Token Ring

Token Ring Failure Alerts

The tables in this section describe the alerts generated by the Token Ring link service provided with Host Integration Server. The alerts are listed in numerical order by alert description and within each description in numerical order by alert ID.

Adapter check error: adapter error alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	1010	Adapter error
Alert ID:	3BA0 3B6D	
Probable causes:	3320	Local Token Ring adapter
Failure causes:	3320	Local Token Ring adapter
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 02):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	02	Data ID: adapter check status
	hh hh	Adapter check status
LAN LCS:		Ring identifier Local individual MAC address

DLC status error: frmr sentinvalid N(R) received

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name.
Alert description:	2100	Software program error
Alert ID:	216D 1033	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F012	Invalid N(r) received
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number

	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR sentreceived i-field exceeded maximum length alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	25AC 0D84	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F023	Received I-field exceeded maximum length
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR sentinvalid/unsupported command/response received alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	28EF 2B5D	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node

	F020	Invalid/unsupported command or response received
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR senti-field received when not permitted alert

Name of subvector	**Hexadecimal value in subvector**	**Description**
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	2C2E 36EA	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F021	I-field received when not permitted
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

Adapter command error: software alert

Name of subvector	**Hexadecimal value in subvector**	**Description**

Alert description:	2100	Software program error
Alert ID:	3EF7 89B7	
Probable causes:	1000	Software program
Failure causes:	1000	Software program
Recommended actions:	30E1	Contact service representative
	32C0	Report the following data:
		Product Set ID index subfield included
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	03	Data ID: adapter return code
	hh	Adapter return code

DLC status error: dm or disk received alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	4B74 26FB	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F01A	DM received
Recommended actions:	3301	If problem persists, do the following:
	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR receivedinvalid N(R) sent alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list		Second name is remote node CP name

Alert description:	2100	Software program error
Alert ID:	83D9 1642	
Probable causes:	2007	LAN LLC communications
	1000	Software program
Failure causes:	1000 F012	Software program Frame reject received: Invalid N(r) sent
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR receivedmax I-field length

Name of subvector	**Hexadecimal value in subvector**	**Description**
Hierarchy/resource list:		Second name is remote node CP name.
Alert description:	2100	Software program error
Alert ID:	8718 0BF5	
Probable causes:	2007	LAN LLC communications
	1000	Software program
Failure causes:	1000	Software program
	F013	Frame reject received: maximum I-field length exceeded
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number

LAN LCS:		Ring identifier Local individual MAC (Media Access Control) address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR receivedinvalid/unsupported commands/response sent alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	8A5B 2D2C	
Probable causes:	2007	LAN LLC communications
	1000	Software program
Failure causes:	1000	Software program
	F010	Frame reject received: Invalid/unsupported command or response sent
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: FRMR received

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	2100	Software program error
Alert ID:	8E9A 309B	
Probable causes:	2007	LAN LLC communications
	1000	Software program
Failure causes:	1000	Software program
	F011	Frame reject received: I-field sent when not permitted

Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

Software program error: adapter interface alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	2100	Software program error
Alert ID:	DA7B C09D	
Probable causes:	3220	Token ring adapter interface
Install causes:	1600	Mismatch between software and microcode
	1601	Incorrect customization image
Recommended actions:	1500	Correct installation problem
	1502	Correct customization parameters
Failure causes:	3220	Token ring adapter interface
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code
LAN LCS:		Ring identifier Local individual MAC address

DLC status error: SABME received for open link station alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name

Alert description:	2100	Software program error
Alert ID:	E65B 0B7F	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
Failure causes:	1023	Communications program in remote node
	F016	SABME received while in ABME
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

Bring-up error alert

Name of subvector	**Hexadecimal value in subvector**	**Description**
Alert description:	3210	Initialization failure
Alert ID:	AB86 8B0B	
Probable causes:	3320	Local Token Ring adapter
Install causes:	1200	Incorrect hardware configuration
	1402	Mismatch between hardware and software configurations
Recommended actions:	1503	Correct configuration
Failure causes:	3320	Local Token Ring adapter
Recommended actions:	30E1	Contact service representative
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code

LAN LCS:		Local individual MAC address

Adapter open error: open failure alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is ring name
Alert description:	3211	Open failure
Alert ID:	016E 5F4E	
Probable causes:	3702	Token ring lobe
	3701	Token ring LAN component
Failure causes:	3712	Local Token Ring lobe
	3701	Token ring LAN component
	2600	Interference
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code
LAN LCS:		Local individual MAC address

Adapter open error: remove command received alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3211	Open failure
Alert ID:	44D1 AD86	
Probable causes:	3705	Token ring remove command received
User causes:	3320	Token ring remove adapter command received
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code

	hh hh	Error code
LAN LCS:		Local individual MAC address

Adapter open error: problem on local lobe alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3211	Open failure
Alert ID:	55BF 3E1C	
Probable causes:	3702	Token ring lobe
Failure causes:	3320	Local Token Ring adapter
	3711	Local Token Ring lobe
	3434	Local lobe cables
Recommended actions:	1009	Try reopening adapter after 30 seconds
	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32C0):
	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code
LAN LCS:		Local individual MAC address

Adapter open error: beaconing detected alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3211	Open failure
Alert ID:	CAF3 C58A	
Hierarchy/resource list:		Second name is ring name
Probable causes:	3703	Token ring fault domain
Failure causes:	3703	Token ring fault domain
Recommended actions:	1009	Try reopening adapter after 30 seconds
	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3101, or 32C0):
	2010	Review link detailed data

	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code

Adapter open error: duplicate address alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3211	Open failure
Alert ID:	D615 A61E	
Probable causes:	3704	Token ring duplicate station address
Install causes:	3704	Token ring duplicate station address assigned
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	07	Data ID: error code
	hh hh	Error code
LAN LCS:		Local individual MAC address

System action: operator caused LAN adapter to be reset alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3211	Open failure
Alert ID:	F7A6 2F59	
Probable causes:	7001	Resources not active
User causes:	3300	Adapter not ready
Recommended actions:	1300	Correct then retry
	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number

	hh	Adapter number
LAN LCS:		Ring identifier local individual MAC address

Network status error: lobe wire fault alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is ring name
Alert description:	3212	Wire fault
Alert ID:	A676 B230	
Probable causes:	3702	Token ring lobe
Failure causes:	3711	Local access unit
	3434	Local lobe cables
	3320	Local Token Ring adapter
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 17):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	17	Data ID: ring status code
	hh hh	Ring status code
LAN LCS:		Local individual MAC address

Network status error: auto-removal alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is ring name
Alert description:	3213	Auto-removal
Alert ID:	EB61 E14F	
Probable causes:	3702	Token ring lobe
Failure causes:	3320	Local Token Ring adapter
	3711	Local access unit
	3434	Local lobe cables
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN

	32C0	Report the error information described by the detailed data subvector (hex values 61 and 17):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	17	Data ID: ring status code
	hh hh	Ring status code
LAN LCS:		Local individual MAC address
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	17	Data ID: ring status code
	hh hh	Ring status code
LAN LCS:		Local individual MAC address

Network status error: Token Ring remove adapter command received alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	3214	Remove adapter command received
Alert ID:	59F3 2622	
Probable causes:	7013	LAN Manager operator
User causes:	7101	Token ring remove adapter command received
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 17):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	17	Data ID: ring status code
	hh hh	Ring status code
LAN LCS:		Ring identifier Local individual MAC address

Network status error: llc remove adapter command received alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	3300	Link error
Alert ID:	5B8F 5BA7	
Probable causes:	2017	LAN LLC communications/remote node
Failure causes:	2107	LAN LLC communications/remote node

	F017	Poll count exhausted
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3103, or 32A0):
	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address
Link Station Data subvector included		

DLC status error: link lost alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	3300	Link error
Alert ID:	9921 0A2B	
Probable causes:	3321	Remote Token Ring adapter
Failure causes:	3321	Remote Token Ring adapter
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address

DLC error: remote station not on local or attached rings alert

Name of subvector	Hexadecimal value in subvector	Description
Hierarchy/resource list:		Second name is remote node CP name
Alert description:	3305	Unable to communicate with remote node

Alert ID:	D91B 9E2D	
Probable causes:	2007	LAN LLC communications
	1023	Communications program in remote node
User causes:	0209	Remote device power off
Recommended actions:	0200	Check power
Failure causes:	1023	Communications program in remote node
Recommended actions:	3301	If problem persists, do the following (available messages correspond to subvectors 2010, 3101, or 32A0):
	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address Remote individual MAC address LAN routing information
LCS configuration:		Remote device address Local device address

Adapter command error: storage capacity exceeded alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	5003	Capacity exceeded
Alert ID:	9C35 13D5	
Probable causes:	0101	Main storage
Failure causes:	0110	Storage control
Recommended actions:	2010	Review link detailed data
	3103	Contact LAN administrator responsible for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 03):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	03	Data ID: adapter return code
	hh	Adapter return code
LAN LCS:		Ring identifier Local individual MAC address

Adapter check error: receive queue overrun alert

Name of subvector	Hexadecimal value in subvector	Description

Alert description:	5012	Receive queue overrun
Alert ID:	14D5 3710	
Probable causes:	1000	Software program
Failure causes:	1000	Software program
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32A0	Report the error information described by the detailed data subvector (hex value 61):
		Product Set ID index subfield included
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
LAN LCS:		Ring identifier Local individual MAC address

Adapter check error: microcode program abnormally terminated alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	6000	Microcode program abnormally terminated
Alert ID:	E1EC 74A7	
Probable causes:	3320	Local Token Ring adapter
Failure causes:	3320	Local Token Ring adapter
Recommended actions:	2010	Review link detailed data
	3101	Contact Token Ring administrator for this LAN
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 07):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
	07	Data ID: error code
	hh hh	Error code
LAN LCS:		Ring identifier Local individual MAC address

Adapter command error: configuration alert

Name of subvector	Hexadecimal value in subvector	Description
Alert description:	8000	Configuration error
Alert ID:	B8AB EF98	
Probable causes:	7001	Local system operator
Install causes:	1501	Incorrect customization parameters

	1400	Mismatch between hardware and software
Recommended actions:	3110	Contact communications systems programmer
	32C0	Report the error information described by the detailed data subvector (hex values 61 and 03):
Detailed data:	61	Data ID: adapter number
	hh	Adapter number
Detailed data:	03	Data ID: adapter return code
	hh	Adapter return code

See Also

Other Resources

Link Alerts for SDLC and Token Ring

Link Statistics

For SDLC (Synchronous Data Link Control) and Token Ring connections, Host Integration Server maintains statistics on link usage and errors. Host Integration Server logs this information when a connection ends or when the counter for a particular error or timeout reaches its maximum value.

This section contains:

Format for Link Statistics

When an SDLC or Token Ring connection ends, or when an error counter reaches its maximum value, Host Integration Server records statistics on link usage. These statistics include information such as the adapter number, the date and time the link was established, and the counts of errors and timeouts. This data is also logged in NMVT (network management vector transport) format if a connection has been specified for carrying NetView data. The data is then sent if that connection and the server that owns the connection are both active.

This section contains:

SDLC Link Statistics

Link statistics are generated when a link is closed or when one of the wrap counters listed later in this section is about to wrap. Whenever a link statistics message is built, all the wrap counters are reset to 0. The cumulative counters are not reset; they provide statistics from the point at which the particular link service was started.

The NMVT generated has the following format:

NMVT header	
41 03 8D 00 00 00 00 00	NMVT Header

Major vector header	
00 69	Length of major vector
00 25	Link statistics major vector

Data link traffic counters subvector	
65 9A	Data Link Traffic Counters subvector
03	DLC type: SDLC
hh	Adapter number (01+04)

Date/time link established	
hh	Date
hh	Month
hh hh	Year
hh	Hour
hh	Minute
hh	Second
hh	Hundredths of a second
hh	Counter

Wrap counters	
hh	Transmit I-frames OK
hh	Transmit I-frames not OK
hh	Retransmit I-frames
hh	Received I-FCS OK
hh	Received I-FCS not OK
hh	Total frames sent

00	Reserved
hh	Lost data frames received
hh	FRMR conditions sent
00	Reserved
hh	CTS drop
00	Reserved
00	Reserved
hh	DSR drop out
hh	Inactivity timeouts
00	Reserved
hh	DMA underruns
00	Reserved
hh	Transmit fail timeouts
00	Reserved

Cumulative Counters

The cumulative counters are the same counters as the wrap counters, with reserved bytes in the same positions.

See Also

Concepts

Format for Link Statistics

Token Ring Link Statistics

Link statistics are generated when a link is closed or when one of the following counters is about to wrap. Whenever a link statistics message is built, all the counters are reset to 0.

The NMVT generated has the following format:

NMVT header	
41 03 8D 00 00 00 00 00	NMVT Header

Major vector header	
00 32	Length of major vector
00 25	Link Statistics major vector

Data link traffic counters subvector	
2E9A	Data Link Traffic Counters subvector
04	DLC type: Token Ring
01 or 02	Statistics type: link counter overflow or adapter counter overflow

Token ring adapter log information and counters	
hh	Adapter number (01+04)
00	Reserved
hh	Line error
hh	Internal error
hh	Burst error
hh	ARI/FCI error
hh	End delimiter
00	Reserved
hh	Lost frame
hh	Receive congestion
hh	Frame copied error
hh	Frequency error
hh	Token error
00 00 00	Reserved

DLC SAP station information	
hh hh hh hh	Count of frames transmitted OK
hh hh hh hh	Count of frames received OK
hh hh hh hh	Count of frames discarded
hh hh hh hh	Lost data
hh hh	Buffer available in SAP

DLC link station information	

hh hh	Count of I-frames transmitted
hh hh	Count of I-frames received
hh	Received I-frames error count
hh	Transmitted I-frames error count
hh hh	T1 timer expired count

See Also

Concepts

Format for Link Statistics

Alerts Used by Applications, NVAlert, and NVRunCmd

Application programs and the NVAlert and NVRunCmd services can generate alerts using the Common Service Verb (CSV) **TRANSFER_MS_DATA**. An application or service can supply subvectors, which are required to build an NMVT, or it can supply the complete NMVT. In both cases, the completed NMVT is logged locally, and may also be sent on the connection designated for NetView if the connection is configured and active. An application or service can also supply user-defined alert data, in which case the data is logged locally but cannot be sent.

This section contains:

Format for Alerts Used by Applications, NVAlert, and NVRunCmd

An application program or the NVAlert and NVRunCmd services can use the CSV **TRANSFER_MS_DATA** to issue alerts. The alerts are logged in a log file that can be viewed using the Windows Event Log service, and may also be sent on the NetView connection.

The data supplied to **TRANSFER_MS_DATA** may be in any of the following formats:

NMVT

The application or service supplies a complete NMVT, including the header information.

ALERT_SUBVECTORS

The application or service supplies the subvectors required for an alert, but without the NMVT header or major vector header. Host Integration Server adds the header information, as described in the next section.

PDSTATS_SUBVECTORS

The application or service supplies the subvectors required for a Problem Determination Statistics NMVT, but without the NMVT header or major vector header. Host Integration Server adds the header information, as described in the next section.

USER_DEFINED

The application or service supplies data in its own format. This data cannot be sent on the connection designated for NetView, but is logged in the same way as the other data formats.

The application or service can also request Host Integration Server to add the Product Set ID subvector (for all data types except USER_DEFINED), the Date/Time subvector, or both to the supplied data. The format of these added subvectors is described in the next section.

For all data types, the data (with added headers and subvectors, if appropriate) is logged locally in NMVT format, whether or not you specified a connection for NetView.

See Also

Reference

Added Headers and Subvectors

Added Headers and Subvectors

The following table lists Added Headers and Subvectors.

NMVT header	
41 03 8D 00 00 00 00 00	NMVT Header

Major vector header	
LL LL	Length of Major Vector
00 00 or 00 25	Alert major vector (for the data type ALERT_SUBVECTORS), or Problem Determination Statistics major vector (for the data type PSTATS_SUBVECTORS)

Product set ID subvector	
4D 10	Product set ID subvector
00	Unused field
34 11	Product ID subvector
0C	Product classification: non-IBM software
08 04	Software product common level subfield
F0 F2	Version identifier: 02
F0 F0	Release identifier: 00
F0 F0	Modification identifier: 00
20 06	Software product common name subfield
C4 C3 C1 61 D4 E2 40 C3 D6 D4 D4 40 E2 C5 D9 E5 D9	Product name: Microsoft Host Integration Server
00 .. 00	Name padded with nulls to 30 characters
09 08	Software product program number subfield
F0 F0 F0 F0 F0 F0 F0	Product program number: zeros
16 11	Product ID subvector
03	Product classification: hardware
13 00	Hardware product ID subfield
00	Format type: undefined
hh...hh	Server: Windows-based computer name

Date/time subvector	
0A 01	Date/Time Subvector
08 10	Local Date/Time subfield
hh	Year two digits
hh	Month
hh	Date
hh	Hours

hh	Minutes
hh	Seconds

See Also

Reference

Format for Alerts Used by Applications, NVAlert, and NVRunCmd

Local Logging of Network Management Data

Data is logged in a Windows Event Log service log file in two ways. First, the data from the alert is logged in a more readable format, using standard log messages. The format is specific to the alert type. For the message numbers used and explanations of the data, see the section in this documentation that relates specifically to that type of alert.

If a connection has been designated for carrying NetView data and the server that owns the connection is active, the alert NMVT containing the data is also logged in NMVT format as it would be sent. The NMVT appears as Message 540, followed by Message 541. Note that this logging occurs before Host Integration Server attempts to send the NMVT. The log is therefore no guarantee that the NMVT is sent successfully.

For more information about local logging of alert information, see the section in this documentation relating specifically to that type of alert.

See Also

Other Resources

Alerts Used by Applications, NVAlert, and NVRunCmd

Network Protocols and Client/Server Communication

This section explains how clients using different local area network (LAN) protocols can communicate with Host Integration Server computers. The section includes an overview of how network protocols are used on clients, and how client logons work. Also included are procedures for checking the setup options for Host Integration Server specified on a client or server. These setup options must be specified correctly in order for the client to communicate with a Host Integration Server computer. You will also find descriptions and illustrations of how clients using each of the possible protocols locate a Host Integration Server computer in the network.

In This Section

Overview of Network Protocols for Clients

Host Integration Server clients can communicate with servers through the following LAN protocols:

- Microsoft Networking (Named Pipes)
- IPX/SPX (the protocol used with NetWare)
- TCP/IP

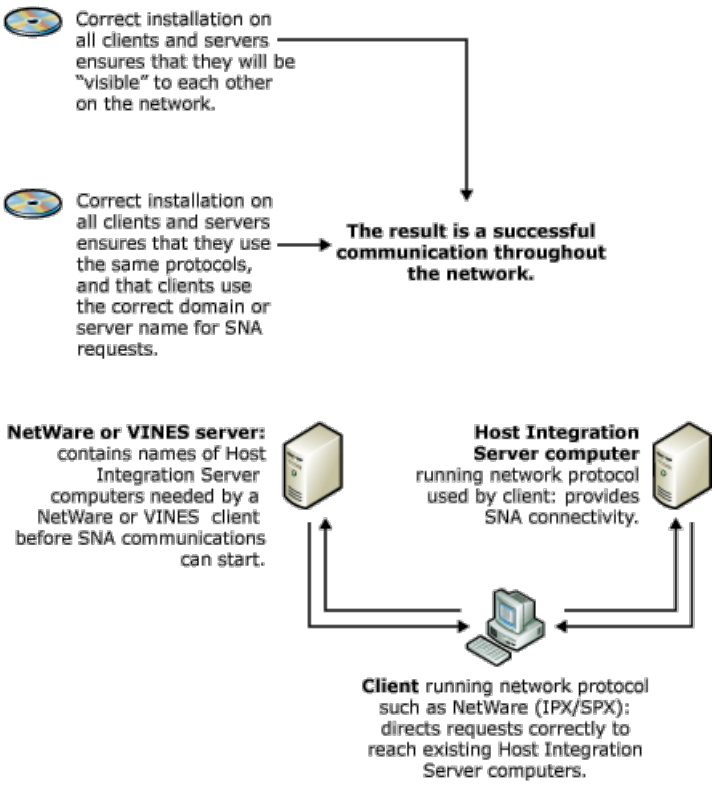
Important
In order for the different client/server protocols to be handled correctly by Host Integration Server, both clients and servers must have the network software and the Host Integration Server software installed correctly. Be sure to follow the installatio n instructions in this section carefully, and follow the installation instructions for other software you are using for example, W indows 2000 Server and the associated network software, Host Integration Server, and network software on clients.

Correct installation of network and Host Integration Server on servers and clients ensures that two essential aspects of communication work correctly:

- The servers and clients are visible to each other on the local area network (LAN). This results when the network software is installed correctly on all affected computers.
- The Host Integration Server computers communicate with clients over the correct LAN protocol, and the clients direct their communication to the correct domain name or, for some clients using Microsoft Networking or TCP/IP, to one or more correct server names. A Host Integration Server client must be set up to use the correct server or domain name or, for Microsoft Networking, set up to locate servers in the local domain. Otherwise, the client will not be able to locate a Host Integration Server computer.

For example, in order for a client to successfully communicate over Novell NetWare, there must be working communication between the client, a NetWare server, or a Windows Server running Microsoft File and Print Services for NetWare and any Host Integration Server computers in the network. Either server provides the client with names of Host Integration Server computers. In addition, for clients running NetWare 4.*x*, bindery emulation must be enabled. The following figure illustrates how correct installation is necessary for successful communication:

Diagram and explanation of how correct installation results in successful communication



For information about how Host Integration Server enables you to view and choose the domain and protocols to be used by Host Integration Server computers and clients, see Important Network Options on a Host Integration Server Computer. For details about how each of the network protocols works on a Host Integration Server client, see the section about that specific network protocol (for example, Clients Using TCP/IP).

In This Section

Client Logons and the Storing of Passwords

Types of Client Logons

Client Logons and the Storing of Passwords

For security reasons, a user at a client computer may have to log on several times before obtaining access through Host Integration Server to a mainframe or AS/400. Users may not always find this to be convenient. The logons are necessary because starting an SNA session requires several kinds of access: access to the Windows domain, access to a Host Integration Server computer, access to the mainframe or AS/400 and, possibly, access to programs on the mainframe or AS/400. An example of such a program is an AS/400 program that uses conversation security when communicating with 5250 emulators. Each layer of access may require an additional logon depending on the operating system and the client/server protocol used on the client.

The Host Security Integration feature of Host Integration Server can reduce or even eliminate these multiple logons on Windows -based networks. If you are not able to take advantage of this feature, users can avoid multiple logons by following the procedures outlined in the following sections.

See Also

Concepts

Types of Client Logons

Types of Client Logons

The number of logons required for establishing an SNA session from a client varies, depending mostly on what client/server protocol the client computer is using. The following list describes the logons:

Clients Using Microsoft Networking

A client using Microsoft Networking must log on to the Windows domain and then must perform any logons required on the mainframe or AS/400. For example, a client using Microsoft Networking and also using a 5250 emulator would log on once to the Windows domain, once to the AS/400 itself, and (in many cases) once to the AS/400 program that communicates with the 5250 emulator.

After clients using Microsoft Networking are logged on to the domain, no additional logon is needed for access to Host Integration Server computers in the domain. That is, Host Integration Server can securely confirm the domain logon without any additional action by the user.

Clients Using Novell NetWare

A client using NetWare must first log on to the NetWare network and then to the Host Integration Server computers. Finally, the client must perform any logons required on the mainframe or AS/400

Clients Using TCP/IP

A client using TCP/IP must log on to the Host Integration Server computers even if the client has already accessed other resources in the Windows domain, and then must perform any logons required on the mainframe or AS/400.

See Also

Other Resources

Overview of Network Protocols for Clients

Important Host Integration Server Network Options

The following two sections illustrate and describe important Host Integration Server network options for a Host Integration Server server or client. These options must be set correctly in order for clients to locate Host Integration Server computers and start sessions. If the options are set incorrectly, clients may use the wrong protocols or may search the wrong locations for Host Integration Server computers.

In This Section

Important Network Options on a Host Integration Server Computer

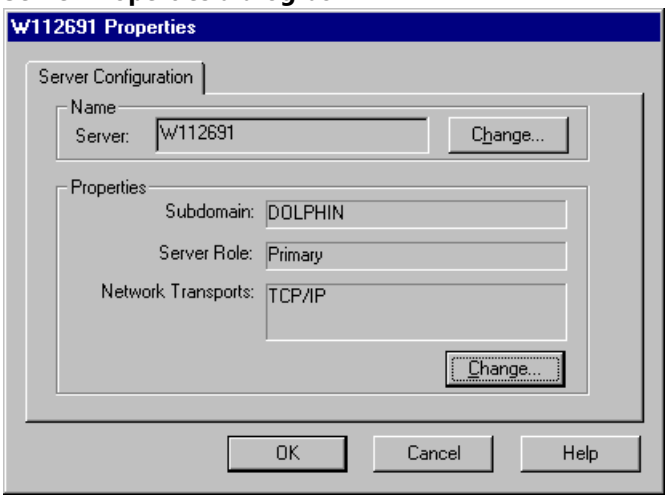
When Host Integration Server network options are specified correctly on a Host Integration Server computer, the server will use the appropriate protocol(s) for communicating with clients. Servers supporting NetWare, TCP/IP, or Microsoft Networking will have the correct domain name so that clients can locate the Host Integration Server computers.

Important
These changes do not take effect until the server is restarted. Do not change these settings while the server has active users.

To view or change client/server protocols and other network options on a Host Integration Server computer

1. In SNA Manager, select a Host Integration Server computer.
2. Right-click a server and then click **Properties**. The **Server Properties** dialog box appears.

   Server Properties dialog box

3. To change the Host Integration Server computer name, click **Change** in the **Name** box. The Host Integration Server computer must be in an off-line state before you can change the name.
4. To modify the client/server protocols and other network options, click **Change** in the **Properties** box, and then click **OK**.
5. Complete the Server SNA Resource wizard.

Note
If the server has TCP/IP installed and will communicate with clients that use TCP/IP, be sure that the selections in the Select Client/Server Protocols dialog box include TCP/IP. If Microsoft Networking (Named Pipes) is selected and TCP/IP is not, the server can communicate through Named Pipes over TCP/IP, a protocol combination that does not recover from error conditions as effectively as native TCP/IP.

If TCP/IP is selected, you can also select Microsoft Networking. In this situation, the Host Integration Server computer will use native TCP/IP with clients that use TCP/IP, and Microsoft Networking with clients that use Microsoft Networking.

See Also

Other Resources

Important Host Integration Server Network Options

Important Network Options on a Client

For Host Integration Server client software, several key options supply the information a client must have for locating Host Integration Server computers. These options specify which protocol the client uses to communicate with Host Integration Server computers if more than one protocol is available on the client, and which domain or server names the client will direct SNA requests to.

When a client makes an SNA request, the client must direct that request to a domain or to one or more Host Integration Server computers. The appropriate way for a client to direct requests depends on the protocol used and the relative location of clients and servers. The following table lists the ways clients direct SNA requests, and the information that will be requested by setup during client installation. For more detail about how a client directs requests and locates Host Integration Server computers, see the section about the appropriate type of client (for example, Clients Using TCP/IP).

Network protocol used on client	**How the client directs SNA requests**	**Information to find out before running client setup**
Microsoft Networking	Either to the local domain (if the Host Integration Server computers are in the same domain as the client), or to one or two specific Host Integration Server computers in a remote domain.	Whether the client is in the same domain as the Host Integration Server computers, and if not, one or two names of Host Integration Server computers.
Novell NetWare (IPX/SPX)	To a specific domain. The Host Integration Server computers must be located in this domain in order for the client to locate them.	The name of the Host Integration Server subdomain in which the Host Integration Server computers are located.
TCP/IP	Either to a domain name or to specific server name, depending on what is specified in the Host Integration Client Mode or Host Integration Server Location dialog box.	Contact your network administrator for additional information regarding a specific IP address for the Host Integration Server computer.

See Also

Other Resources

Important Host Integration Server Network Options

Adjusting Clients Running Windows for Workgroups

Clients running Windows for Workgroups have a very flexible set of options for network operation. As a result, these clients may require specific adjustments in order to communicate with Host Integration Server computers.

Clients running Windows for Workgroups and using Microsoft Networking (not other network software) must use domain and password settings that coordinate with settings in Host Integration Server Setup. For example, if the client's workgroup name does not match the server's domain name, options in Host Integration Server Setup must be set accordingly.

In This Section

Domain and Password Settings for Clients Running Windows for Workgroups

This section applies only to clients running Windows for Workgroups and using Microsoft Networking to communicate with Host Integration Server computers. Such clients must be configured so that any domain names, workgroup names, local passwords, and domain passwords all work together smoothly.

In This Section

Domain Settings with Windows for Workgroups

For domain names and workgroup names, use one of the following approaches:

- **Workgroup name differs from Host Integration Server domain name.**

  For clients where the workgroup name must be different from the server's domain name (the name of the Windows domain), in Host Integration Server Client Setup, you must use the Remote Domain setting. That is, in the Host Integration Server Location dialog box, select Remote Domain, and supply the name of one or two Host Integration Server computers to which the client will send requests.

  To view a client's workgroup name, in Control Panel, double-click the Network icon. The name is listed in the resulting dialog box.

- **Workgroup name matches Host Integration Server domain name.**

  For clients where the workgroup name is the same as the server's domain name (the name of the Windows domain), in Host Integration Server Client Setup, you can use the Local Domain setting. That is, in the Host Integration Server Location dialog box you can select Local Domain. When the client requests an SNA session, the request will reach the domain that has the same name as the client's workgroup.

  To view a client's workgroup name, in Control Panel, double-click the Network icon. The name is listed in the resulting dialog box.

See Also
Tasks
Password Settings with Windows for Workgroups

Password Settings with Windows for Workgroups

For Windows for Workgroups passwords and domain passwords, use one of the following approaches:

- **Enable logon to the Windows domain automatically at startup.** You can configure Windows for Workgroups to log the user on to the Windows domain automatically at startup.To do this, first make sure that a user account has been created for the user in the Windows domain, and obtain the assigned user name and password.

To configure Windows for Workgroups to logon to the Windows domain automatically at startup

1. On the client computer, from the Control Panel, double-click the **Network** icon and then click **Startup** .
2. In the **Startup Settings** dialog box, under **Options for Enterprise Networking**, find the check box labeled **Log On to Windows or LAN Manager Domain**. Select this check box, and type the domain name in the appropriate box.
3. Choose **OK** repeatedly until the dialog boxes are closed.

The next time Windows for Workgroups is restarted, the user will be prompted for both the local password and the domain password, and must type them correctly. Thereafter, only the local password is needed at startup, because the domain password is encrypted and stored (to be "unlocked" by the local password).

After domain logon has been set up, it is important for the user to understand the difference between the local Windows for Workgroups password and the actual domain password. The local Windows for Workgroups password unlocks the encrypted password list that has been stored for that user on that computer. This password list includes a copy of the domain password if the preceding configuration steps have been taken. The actual domain password is the password for that user's account on the Windows domain, as recorded on Windows servers that are domain controllers.

The user's domain account may be set up to require periodic changing of the password. If the domain password expiration date is getting close, when the user logs on to the domain, pop-up warnings will appear on the user's computer. Note that the user must try to log on in order for the pop-up warnings to appear. At this point, the user needs to understand the different buttons used for changing passwords with Windows for Workgroups. After choosing the Network option from the Control Panel, click the Password button. This only changes the password to unlock the user's local password list; it does not change the domain password.

To change the domain password, the user must choose the Startup button followed by the Set Password button. When the user changes the domain password using the Set Password button, the domain password is also changed in the local password file.

Make sure that users understand how to use these buttons correctly to change their local Windows for Workgroups password and to change their Windows domain password.

- **Log on only as needed, by using the net logon command.** If the user does not log on to the Windows domain at startup, the following command can be used before starting an SNA session. At the MS-DOS prompt (from within Windows for Workgroups), type

  net logon *username* **/domain:***domainname*

  After typing the command, the user is prompted for the domain password.

See Also
Concepts
Domain Settings with Windows for Workgroups

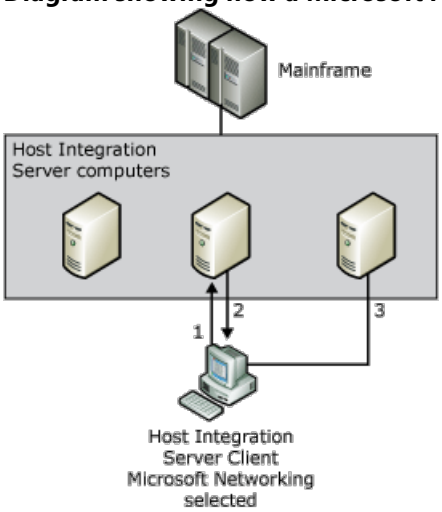
Details about How Clients Use Protocols

The following sections describe how each type of client works with Host Integration Server. The clients are listed according to the protocol used: Microsoft Networking (Named Pipes), Novell NetWare (IPX/SPX), and TCP/IP.

In This Section

Clients Using Microsoft Networking (Named Pipes)

Diagram showing how a Microsoft Networking client can connect to the mainframe



A Microsoft Networking (Named Pipes) client uses the following procedure for connecting to the mainframe:

1. You can choose between two methods by which a client searches for Host Integration Server computers.

   - Active Directory
   - Sponsor connections, locating servers by SNA subdomain or by server name.

   Client locates servers by subdomain means that the client locates the server through broadcasts in the local subdomain. Therefore, the client must be in the same physical network as the Host Integration Server computers, and must not be separated from the servers by a router.

   Client locates servers by name means that the client searches for Host Integration Server computers by name, and therefore need not be on the same side of any routers as the Host Integration Server computers. When connecting with Named Pipes over TCP/IP, and there is a router separating the Host Integration Server computer and the client, the client computer requires a method of resolving NetBIOS names to an IP address. A local LMHOSTS file or a Windows Internet Name Service (WINS) server may be used to perform this resolution.

Note
If Windows for Workgroups 3.11 is being used and the Microsoft IPX/SPX compatible transport is installed, add **Direct host=no** to the [network] section of SYSTEM.INI file on the Windows for Workgroups 3.11 client computer, and then r eboot the computer.

   If this is a routed network, Named Pipes is not recommended. TCP/IP is the recommended interface for this environment.

2. The Host Integration Server computer responds with a list of available Host Integration Server computers that are available to get a 3270 or APPC session with. This special connection is called the sponsor connection.

3. Finally, the client attempts to connect to each Host Integration Server computer in the list until a server is found that can handle the 3270 or APPC request.

See Also

Tasks

Microsoft Networking (Named Pipes) Errors

Microsoft Networking (Named Pipes) Errors

NAMED PIPES 5
Causes

One of the following has occurred:

- Access has been denied trying to create a connection to the Host Integration Server computer. The client/server interface connects to the Host Integration Server computer just like any other local area network (LAN) connection. It must be validated by Windows as a valid user in the domain.

- The Host Integration Server computer is unreachable. If using TCP/IP this could mean that the name to IP address resolution has failed.

NAMED PIPES 2
Causes

The client has connected to the Host Integration Server computer, but the SNABase service is not running on that machine. Start the SNABase service on the Host Integration Server computer to which you are trying to connect.

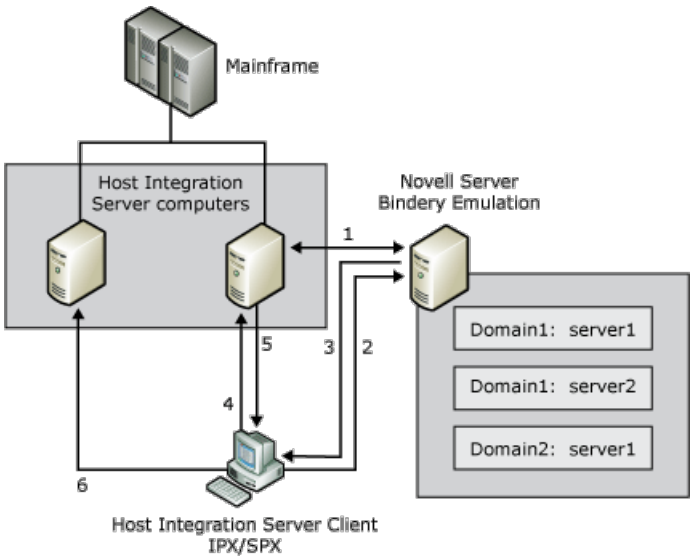
NAMED PIPES 121
Causes

The client is configured for **Client locates servers in an Host Integration Server 2009 Subdomain** and therefore, is sending out broadcast messages to discover a Host Integration Server computer. These broadcast messages will typically not be passed by a router that may be between your client and the server. In the client configuration select **Client locates server by name** and specify an IP address or computer name of one or more Host Integration Server computers.

See Also
Concepts
Clients Using Microsoft Networking (Named Pipes)

Clients Using NetWare (IPX/SPX)

The following diagram shows how a client using NetWare can connect to the mainframe.



Novell NetWare (IPX/SPX) client computers use the following procedure for connecting to the mainframe:

1. At startup time, all Host Integration Server computers register their subdomain name and computer name with the NetWare Bindery. The subdomain name is configured in Setup for Host Integration Server. The computer name used is the Windows computer name.

2. Sometime later, a client requests an initial SNA session (3270 or APPC) by sending the request to the NetWare server requesting a list of registered Host Integration Server computers.

3. In response, the NetWare server sends the client the names of the Host Integration Server computers.

4. The client then selects one of the names from this list of Host Integration Server computers at random, and attempts to get a LAN connection with one of the Host Integration Server computers. The client must be configured for the same subdomain name as the Host Integration Server computers.

5. The Host Integration Server computer responds with a list of available Host Integration Server computers that are available to get a 3270 or APPC session with. This special connection is called the *sponsor connection*.

6. Finally, the client attempts to connect to each Host Integration Server computer in the list until a server is found that can service the 3270 or APPC request.

See Also

Tasks

Checklist for Clients Using NetWare (IPX/SPX)

NetWare Errors

Checklist for Clients Using NetWare (IPX/SPX)

When troubleshooting client problems, check that all of the following conditions have been met:

For Bindery

- **NetWare Servers**

  The NetWare server is properly installed and running, so that the bindery service is available.

- **Host Integration Server Computers**

  The Host Integration Server computer has registered its name in the bindery on the NetWare server. Several elements are required for this. The first element happens automatically: the installation of the NWLink IPX/SPX Compatible Transport on Windows. The second is matching the frame-type with the NetWare Server. For instance, if 802.2 is configured on the NetWare Servers, then the same frame type should be configured in the properties for the NWLink IPX/SPX Compatible Transport on Windows. Auto Frame Type is recommended.

  The correct subdomain name must be typed in the **Host Integration Server 2009 Subdomain Name** dialog box. The name is registered (along with the server name computer name) on the NetWare server. Only if the NetWare server is provided with the correct subdomain name can it respond to client requests and provide the clients with Host Integration Server names.

  Also, for the Host Integration Server computer to register its name on the NetWare server, Host Integration Server must be installed to work with IPX/SPX. This means that in Setup, the Novell NetWare (IPX/SPX) option must be selected in the Select Client/Server Protocols dialog box.

- **IPX Routers**

  In addition, for Bindery, any IPX routers between the two servers must be configured to allow NetWare Service Advertising Protocol broadcasts (SAP broadcasts)() to flow. IPX routers must propagate sockets 84C8 and 84C9 to support the communication needed by Host Integration Server computers and clients.

  To discover whether any Host Integration Server computers have been registered with NetWare Bindery, use the NetWare Rconcole utility against any NetWare Server that is using Bindery. For instance, if there are two Host Integration Server computers (Server1, Server2) that have successfully registered themselves with the NetWare Bindery and both servers are configured to be in a subdomain called Domain1, the entry in the NetWare Rconsole utility would be:

  - DOMAIN1!Server1
  - DOMAIN1!Server2

For Client

- Ensure that the client is using the same subdomain name as the Host Integration Server computers you want to connect to and has NWLink installed.

- **NDS Specific**

  The same concepts for Bindery apply to NetWare Directory Service (NDS), which the exceptions:

- **Host Integration Client**

  The NetWare Directory Service (NDS) logon credentials on the Host Integration Server computer must have the right to add an entry into the NetWare NDS tree structure. If this fails, there will be an event logged to the Windows Application Event Log.

The NDS Context and Tree Name must be provided by the client and server configurations in addition to the subdomain name in order for a client to connect successfully.

- **NetWare Servers**

  If using NetWare Directory Services (NDS), Host Integration Server requires NetWare version 4.11 or later. Earlier versions of NetWare are supported through the bindery emulation feature of NetWare version 4.0 or later.

Note
If NDS is selected on the client and an attempt to get a sponsor connection fails, it will automatically try Bindery.

See Also
Tasks
NetWare Errors

NetWare Errors

- **BINDERY 250**

  The Host Integration Server computer has contacted a NetWare Server and has enumerated a list of available Host Integration Server computers in the client's subdomain. However, the client cannot connect to any of the Host Integration Server computers in this list. There may be a network problem preventing a connection. Ensure that the same frame type is being used on the client and the other Host Integration Server computer.

- **BINDERY 252**

  The client has found that there are no Host Integration Server computers in this subdomain registered with the NetWare Bindery. Ensure that the clients and servers are using the same subdomain name and that SNABase service on the Host Integration Server computers has been started and registered with Bindery. To check whether SNABase has been registered with the NetWare Bindery use the NetWare Rconsole utility. For instance, if the Host Integration Server computer is configured to be in a subdomain called Seattle, and the computer name of the Host Integration Server computer is SNAS1, then the entry in Rconsole would be:

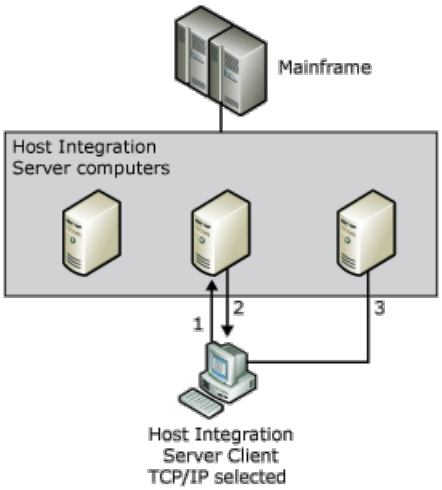
  SEATTLE!SNAS1

- **NDS 64935**

See Also
Tasks
Checklist for Clients Using NetWare (IPX/SPX)

Clients Using TCP/IP

The following figure shows how a TCP/IP client computer can connect to the mainframe.



A TCP/IP client computer uses the following procedure to connect to the mainframe:

1. With TCP/IP, you can choose between two methods by which a client searches for Host Integration Server computers.

   - **Client locate servers by subdomain** means that the client locates the server through broadcasts in the local subdomain. Therefore, the client must be in the same physical network as the Host Integration Server computers, and must not be separated from the servers by a router.

   - **Client locates servers by name** means that the client searches for Host Integration Server computers by name or IP address, and therefore need not be on the same side of any routers as the Host Integration Server computers. When using a name (instead of an IP address) and there is a router separating the Host Integration Server computer and the client, the client computer requires a method of resolving a name to an IP address. A local LMHOSTS file or a Windows Internet Name Service (WINS) server may be used to perform this resolution.

2. The Host Integration Server computer responds with a list of available Host Integration Server computers that are available to get a 3270 or APPC session with. This special connection is called the sponsor connection.

3. Finally, the client attempt to connect to each Host Integration Server computer in the list until a server is found that can service the 3270 or APPC request.

Note that before attempting to establish communication involving Host Integration Server, you can use the TCP/IP utility called **ping** to verify that contact can be established between client and server. This can help you identify basic TCP/IP problems, such as difficulties with name resolution or with routers. For information about the **ping** utility, see the TCP/IP documentation.

Note
Ensure that the logged-on user has sufficient user rights to access the Host Integration Server computer across the network. As with any other LAN connection, this is validated through the Windows Domain model.

See Also
Tasks
TCP/IP Errors

TCP/IP Errors

TCP/IP 11004
Causes

Your client is configured for Remote client locates servers by name, and you have selected a computer name instead of an IP address. The computer name to IP address resolution has failed, or the IP address resolved cannot be found on the network. Try inputting an IP address instead of a computer name into the client configuration. If this works, then there may be a problem with your local lmhosts file, DNS Server or WINS when using a computer name in the configuration. If this does not work, try pinging the address.

TCP/IP 10061
Causes

The SNABase service on Host Integration Server computer is not running or the IP address cannot be resolved, either because this Server is not on the network, the IP address is incorrect, or there is a router or network problem. Try pinging the address.

TCP/IP 121
Causes

The client is configured for Local Client locates servers in a Host Integration Server Subdomain and therefore, is sending out broadcast messages to discover a Host Integration Server computer. These broadcast messages will typically not be passed by a router that may be located in between your client and the server. Go into the client configuration and select Client locates server by name and specify an IP address or computer name of one or more Host Integration Server computers.

See Also
Concepts
Clients Using TCP/IP

Error Messages

The messages recorded in the event log for Host Integration Server are now available in a database. The Message Database file, included on the Host Integration Server CD-ROM at **\Documentation\Message Database\Snamsg.mdb**, requires Microsoft Access 97 or later for viewing. This database enables you to design your own queries and reports to gather and analyze information regarding Host Integration Server events. The message database includes the message identifier, severity level, symbolic message name, source, message, and an explanation where appropriate.

Some errors and events have additional information, such as debugging suggestions or recommended actions, available in the Error Message Database (EMDB). You can access the EMDB through the link provided in the error and event log. This link will take you directly to the error in the EMDB.

You can also access the entire EMDB at the following location:

http://go.microsoft.com/fwlink/?LinkId=33506

Finally, you can view events using the Windows Event Log service on the **Tools** menu in SNA Manager. Once you have the Windows Event Log service running, double-click an event to display the **Event Detail** dialog box. Then use the **Event Detail** dialog box to see more information about a selected event.

See Also
Other Resources
Administrator's Reference

Command-Line Interface

In addition to the graphical user interfaces provided by Host Integration Server Setup and the SNA Manager, Host Integration Server offers a command-line interface. The command-line interface can be useful in certain situations, such as when you want to view a configuration setting quickly without starting the graphical interface, or when you want to store configuration commands in a command file so that they can be carried out easily in the future.

Before using the command-line interface, you must install Host Integration Server and run the SNA Manager at least once. This initializes important elements of the Host Integration Server configuration that cannot be controlled with the command-line interface.

You can use the command-line interface to modify an offline configuration file (as well as a regular on-site configuration file). However, when you work with an offline configuration file, validation of server names, link names, and user names cannot take place, and errors may result. For information about specifying the name of the configuration file to modify with the command-line interface, see Specify the Subdomain Configuration File.

Other actions are also not available from the command-line. For example, you cannot use the command-line interface to configure ranges of LUs.

Certain Host Integration Server commands must be used with great care if the **/add** option is used with them. With these commands **snacfg server**, **snacfg link**, and **snacfg user** the information you type must match existing information in your system or on the Windows domain.

Important
If you specify server names, link service names, adapter properties, or user names by using the /add option with snacfg server, snacfg link, or snacfg user, these names or properties must match existing names and properties in your installation, or a nonfunctioning configuration may result. To protect against errors with these commands, use the SNA Manager instead.

In This Section

Snacfg Reference

Linkcfg Reference

Kerberos Support

In addition to NTLM, Kerberos support is now available. To use Kerberos support, it is necessary to set a Service Principal Name (SPN) for each server in the subdomain. You can do this through the command line as follows:

```
setspn -a hisservice/servername serviceaccount
```

An icon will appear in the status bar to verify that the system is using Kerberos. Kerberos support is compatible with previous versions of Host Integration Server.

Snacfg Reference

The following sections reference specific areas of Snacfg.

This section contains:

- Snacfg Server
- Snacfg TN3Server
- Snacfg TN5Server
- Snacfg TN3Session
- Snacfg TN5Session
- Snacfg TNIPID
- Snacfg User
- Snacfg Workstation
- Snacfg Error Messages

Task Order

Just as with the SNA Manager, configuration tasks must be carried out in a certain order with the command-line interface. For example, before configuring an LU, you must configure the connection that the LU will use. You can vary the interface you use for each configuration task, as long as you carry out the tasks in order, and as long as you have already installed Host Integration Server with the Host Integration Server Setup and then run the SNA Manager. For example, you can carry out tasks using the SNA Manager, then the command-line interface, then the SNA Manager again, as long as the tasks are done in the correct order.

The following table shows the order in which configuration tasks must be carried out, and the command used to carry out the task.

Task Order for Configuration Tasks

Configuration task (in order)	**Command that carries out task**
Add a server to the configuration	**snacfg server***
Configure link service(s) for the server(s)	**snacfg link***
Configure connection(s) for the server(s)	**snacfg connection**
Configure LU(s) for the connection(s)	**snacfg appcllu**, **snacfg appcrlu**, **snacfg lu**, **snacfg lua**, or **snacfg lud**
Create LU pool(s) (optional)	**snacfg pool**, **snacfg poola**, or **snacfg poold**
Assign LUs to pool(s) (optional)	**snacfg lu**, **snacfg lua**, or **snacfg lud**
Add users to the list used by Host Integration Server	**snacfg user***
Assign LU(s) or LU pool(s) to users	**snacfg user**

* It is recommended that you use the SNA Manager, not the command-line interface, for these tasks. Any errors in the typing of commands for these tasks can result in a nonfunctioning configuration.

See Also

Other Resources

Snacfg Reference

Help with the Command-Line Interface

To see the syntax for a particular **snacfg** command, follow the command with the **/?** option.

For example, to get a listing of the words that can follow **snacfg** in the command-line, type

```
```

To get a listing of the options that can follow **snacfg connection**, type

```
```

See Also

Other Resources

Snacfg Reference

Specify the Subdomain Configuration File

Within a Host Integration Server command (a **snacfg** command), you can specify the path of the subdomain configuration file to access, or you can omit the path. If you omit the path, Host Integration Server attempts to access the configuration file in the normal location on the local system: \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG. To specify the path of the configuration file to access, type the **snacfg** command with the following syntax:

```
configpath command options
```

That is, follow the **snacfg** command with a space, a pound sign, then the configuration path, and then any additional command syntax.

For example, to view a listing of the connections stored in Program Files\Host Integration Server\SYSTEM\CONFIG\BACKUP.SNA, type

```
```

See Also

Other Resources

Snacfg Reference

Use a Command File

If you want to run a series of Host Integration Server configuration commands, you can remove the word **snacfg** from each command, place the new commands in a file called a command file, then use a single **snacfg** command to run the entire command file. This is similar to the way a batch file works; however, a command file opens and closes the configuration file fewer times than a batch file. When a command file is run, the configuration file is opened only once, at the beginning. Then all the commands are carried out, and the configuration file is closed. In contrast, when a batch file containing **snacfg** commands is run, the configuration file is opened and closed multiple times, once for every command in the file.

When creating a command file, do not include the following:

- The word **snacfg**
- A path for a configuration file
- A command path for another command file
- A backslash inside the text string for a comment

Also, you can include long, multiline commands in a command file by ending lines with a backslash (\ ). The backslash indicates that the string in the next line should be appended to the current command.

There are two steps for using a command file. First, create the file, either by typing the configuration commands into a plain text file, or by using the **/print** option as described in the next section. Then run the command file from the command prompt by typing a line with the following syntax:

```
        [configpath]commandpath []
```

In the preceding syntax line, *configpath* is the path of the configuration file on which commands should be carried out; precede this path with the **#** symbol. Similarly, *commandpath* is the path of the command file; precede this path with the **@** symbol. Use the **/v** (verbose) option to cause all informational messages (not just error messages) to be displayed when the command file is running. Without the **/v** option, only error messages are displayed.

For example, to run a series of commands that result in a listing of the links and connections in a configuration file, create a file called SNA_CMD1.TXT, containing the following lines:

```
link /list
connection /list
```

See Also
Other Resources
Snacfg Reference

Create a Snacfg Command File from a Configuration File

To create lines for a **snacfg** command file, you can type them, or you can generate them by using the **/print** option. The **/print** option accesses an existing configuration file and generates the command-line(s) required to add an individual resource or (depending on syntax) the entire configuration file.

The output generated by the **/print** option does not contain the word **snacfg**. This means that the output can be included in a **snacfg** command file.

See Also

Other Resources

Snacfg Reference

General Syntax for the /print Option

The syntax lines in this section and the next section show ways to use the **/print** option. By default, the output is sent to the screen. To capture the output in a file, redirect it in the standard way, by adding a greater-than sign (>) to the end of the command, followed by the name of the file in which you want to capture the output.

With all the syntax lines, if the source configuration file is not in the default path \Program files\Host Integration Server\SYSTEM\CONFIG\COM.CFGthen for **#***configpath*, substitute a path, preceded by the pound sign (**#**). Do not type the square brackets. After the greater-than sign (**>**), type the name of the command file you want to create. (For information about using the greater-than sign or other methods of redirection, see your Windows documentation.)

The general syntax for the **/print** option is:

```
        [configpath] [resource [location]resourcename] cmdfile.ext
```

For *resource*, substitute the second word of a **snacfg** command (for example, **appcllu**, **connection**, or **server**). For *location*, if needed, substitute the server or connection name that uniquely locates a resource; for *resourcename*, substitute the name of the resource.

See Also
Other Resources
Snacfg Reference

Examples of Syntax for the /print Option

The following examples illustrate the use of the **/print** option.

- To create a command file that can recreate an entire configuration file, type a command of the following form:

```
[configpath] cmdfile.ext
```

- To create a command file from a particular connection in an existing configuration, type a command of the following form, substituting the name of the connection for *connectionname*:

```
[configpath] connectionnamecmdfile.ext
```

- To create a command file from a particular 3270 LU in an existing configuration, type a command of the following form, substituting the name of the LU for *luname*:

```
[configpath] lunamecmdfile.ext
```

After generating **snacfg** command files, you can modify them and then use them like any other **snacfg** command file.

See Also
Other Resources
Snacfg Reference

Use the /print Option

The **/print** option can make it easier to carry out repetitive configuration actions, if you have a detailed understanding of Host Integration Server configurations and of the **snacfg** command. Here are some ways of using the **/print** option:

Creating a new configuration

You can generate a **snacfg** command file corresponding to an entire configuration file, modify the command file, and use it to create a new configuration file for another subdomain or site. To generate the commands for an entire configuration file, use the syntax shown in the first example in the previous section; that is, omit all **snacfg** modifiers (such as **connection** or **lu**).

Creating a template to use for expanding one or more configurations

You can generate a **snacfg** command file that corresponds to a useful element of an existing configuration (for example, a 3270 LU in a configuration). Then you can modify the file and use it to add one or many similar elements to an existing configuration. Such a command file acts as a template for the element that it contains.

Modifying a configuration

You can generate a **snacfg** command file that corresponds to some part of an existing configuration (for example, an 802.2 connection in a configuration), modify the command file, and use it to modify the configuration file from which it came. This involves removing the **/add** option from lines in the command file, and making other changes that require a detailed understanding of the commands in the file.

See Also

Other Resources

Snacfg Reference

Display the Contents of a Configuration File

You can create a display of the entire contents of a configuration file by using the **/display** option with the following syntax:

```
[configpath]
```

When you use the **/display** option, all the resources in the configuration file are displayed. For each resource, the display is the same as that from typing **snacfg** *resource resourcename*, where *resource* is the second word of a **snacfg** command (for example, **appcllu**, **connection**, or **server**), and *resourcename* is the name of the corresponding resource. (The exception to this is that the display of the "diagnostic" resource is the same as that from **snacfg diagnostic /list**.)

See Also

Concepts

Use the /print Option

General Syntax for the /print Option

Snacfg APPCLLU

Purpose

Allows you add, delete, modify, or view a local APPC LU. Also allows you to view the command that would create a specified local APPC LU.

Note
Configuration settings specified with snacfg appcllu correspond to local APPC LU settings configured with the SNA Manager.

Syntax

```
snacfg [#configpath] appcllu /list
```

Recommended Syntax

```
        [configpath]  servernameLUalias
[configpath]  LUalias servername [options]
[configpath]  servernameLUalias [options]
[configpath]  servernameLUalias  [configpath]  servernameLUalias
```

Other Available Syntax

```
        [configpath]  [configpath]  LUalias
[configpath]  LUalias [options]
[configpath]  LUalias  [configpath]  LUalias
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\System\CONFIG\COM.CFG.

/list

Generates a list of configured local APPC LUs.

servername **:** *LUalias*

Specifies the server name and LU alias of the local APPC LU on which to carry out actions. The server name should be in the format machine_name or \\machine_name\snaservr (for specifying the primary node on the machine) and \\machine_name\snasrv02 (or snasrv03, snasrv04, etc.) for specifying the secondary nodes on the machine.

It is recommended that *servername***:** be included in **snacfg appcllu** commands (other than **/add** commands) that include *LUalias*. Without *servername***:**, if there is more than one local LU called *LUalias* in the subdomain, it is difficult to predict which of these LUs will be affected by the command. The **snacfg appcllu** command does not necessarily default to the local server if *servername* is omitted.

See the following paragraphs for details about characters permitted in the LU alias.

If no options are specified after *LUalias*, the configuration settings, partner LUs, and modes are displayed for the specified LU.

LUalias

Specifies the LU alias of the local APPC LU on which to carry out actions. See the previous paragraphs and the syntax lists for recommendations about using the server name with the LU alias.

The LU alias can be from one through eight characters long, and can contain alphanumeric characters and the special characters %, $, #, and @. Lowercase letters are converted to uppercase. For a local APPC LU, the LU alias must be unique on the server.

Note

LUalias is used as the default LU name if **/luname:***text* is not specified.

If no options are specified after *LUalias*, the configuration settings, partner LUs, and modes are displayed for the specified LU.

/add

Adds a local APPC LU called *LUalias*. To configure the LU, either specify other options after **/add**, or specify configuration options in additional **snacfg appcllu** commands (using the same *LUalias*).

/connection: *conn-name*

When adding or modifying a dependent local APPC LU, this option is used to specify the LU's connection assignment.

/delete

Deletes the LU called *LUalias*.

/print

Causes the display of the **snacfg** command that would create the specified local APPC LU. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file.

Options for Local APPC LUs

/server: *servername*

Specifies the server to which to assign or move the APPC LU. When **/add** is used, this option is required. The server name should be in the format machine_name or \\machine_name\snaservr (for specifying the primary node on the machine) and \\machine_name\snasrv02 (or snasrv03, snasrv04, etc.) for specifying the secondary nodes on the machine.

/lunumber: *value*

Specifies the LU number. If an LU number of 0 is specified, the LU is configured as an independent local APPC LU.

/netname:" *text*"

Specifies a name for the network of this LU. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @.

If **/netname:***text* is not specified, the network name of the Host Integration Server on which the LU is located is used as the default.

Note

/luname:*text* also has a default (the LU alias). Therefore, the fully qualified LU name (network name plus LU name) can potent ially be created by default, if the LU alias and local network name are configured appropriately. A fully qualified LU name is r equired for an APPC LU.

/luname:" *text*"

Specifies the LU name. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. For a local APPC LU, the fully qualified LU Name (Network Name plus LU Name) must be unique on the server.

If **/luname:***text* is not specified, *LUalias* is used as the default LU name.

Note

/netname:*text* also has a default (the network name of the server on which the LU is located). Therefore, the fully qualified LU name (network name plus LU name) can potentially be created by default, if the LU alias and local network name are configu red appropriately. A fully qualified LU name is required for an APPC LU.

/comment:" *text*"

Adds an optional comment for the LU. The comment can contain as many as 25 characters; enclose the comment in quotes.

/autopartner:{ yes| no }

Specifies whether this LU will automatically be partnered with other APPC LUs.

If automatic partnering is left unspecified, the default is **yes**.

/defaultpool:{ yes| **no }**

Specifies whether this LU will be in the default outgoing local APPC LU pool. This pool makes LUs available for invoking TPs that do not specify a local LU.

/impremotelu: *remoteLUname*

Specifies an existing remote LU to be used as an implicit incoming remote LU for the local LU.

/tptimeout: *value*

Specifies the number of seconds that Host Integration Server should wait for the invokable TP to respond to a start request from the invoking TP.

/addpartner: *LUalias* **,** *mode*[**,***connection*]

Partners the local LU with the specified LU and the specified mode. Both *LUalias* and *mode* must exist before they can be specified as partners. If *LUalias* specifies a remote LU that is not unique on the server, the connection used by the remote LU must also be specified; otherwise, Host Integration Server will randomly choose one of the remote LUs called *LUalias* to act on.

Only one **/addpartner** option can be used in each command.

/delpartner: *LUalias* **,** *mode*[**,***connection*]

Deletes the pair listing that includes this local LU, the specified partner LU, and the specified mode. (Does not delete any LUs or modes themselves.) If *LUalias* specifies a remote LU that is not unique on the server, the connection used by the remote LU must also be specified; otherwise, Host Integration Server will randomly choose one of the remote LUs called *LUalias* to act on.

Only one **/delpartner** option can be used in each command.

/syncpoint:{ yes | no }

Specifies if this Local APPC LU provides LU 6.2 SyncPoint support.

/clientname: "*text*"

Specifies client name if SyncPoint support is enabled.

See Also

Other Resources

Snacfg Reference

Snacfg APPCRLU

Purpose

Allows you add, delete, modify, or view a remote APPC LU. Also allows you to view the command that would create a specified remote APPC LU.

Note
Configuration settings specified with snacfg appcrlu correspond to remote APPC LU settings configured with the SNA Manager.

Syntax

```
snacfg [#configpath] appcrlu /list
```

Recommended Syntax

```
        [configpath]  connectionnameLUalias
[configpath]  LUalias connectionname [options]
[configpath]  connectionnameLUalias [options]
[configpath]  connectionnameLUalias  [configpath]  connectionnameLUalias
```

Other Available Syntax

```
        [configpath] LUalias [configpath] LUalias [options]
[configpath] LUalias [configpath] LUalias
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured remote APPC LUs.

connectionname **:** *LUalias*

Specifies the connection and LU alias of the remote APPC LU on which to carry out actions. You should include *connectionname***:** in **snacfg appcrlu** commands (other than **/add** commands) that include *LUalias*. Without *connectionname***:**, if there is more than one remote LU called *LUalias* in the subdomain, it is difficult to predict which of these LUs will be affected by the command. The **snacfg appcrlu** command does not necessarily default to a connection on the local server if *connectionname* is omitted.

See the following paragraphs for details about characters permitted in the LU alias.

If no options are specified after *LUalias*, the configuration settings, partner LUs, and modes are displayed for the specified LU.

LUalias

Specifies the LU alias of the remote APPC LU on which to carry out actions. See the previous paragraphs and the syntax lists for recommendations about using the connection name with the LU alias.

The LU alias can be from one through eight characters long, and can contain alphanumeric characters and the special characters %, $, #, and @. Lowercase letters are converted to uppercase. For a remote APPC LU, the LU alias must be unique on the connection, and must not match that of a local LU on that server.

If no options are specified after *LUalias*, the configuration settings, partner LUs, and modes are displayed for the specified LU.

/add

Adds a remote APPC LU called *LUalias*. To configure the LU, either specify other options after **/add**, or specify configuration options in additional **snacfg appcrlu** commands (using the same *LUalias*).

/delete

Deletes the LU called *LUalias*.

/print

Causes the display of the **snacfg** command that would create the specified remote APPC LU. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. See the information about command files earlier in this section.

Options for Remote APPC LUs
/connection: *connectionname*

Specifies the connection to which to assign or move the APPC LU. When **/add** is used, this option is required.

/netname:" *text*"

Specifies a name for the network of this LU. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @.

If **/netname:***text* is not specified, the remote network name configured for the connection supporting this LU is used as the default.

Note
/luname:*text* also has a default (the LU alias). Therefore, the fully qualified LU name (network name plus LU name) can potentially be created by default, if the LU alias and remote network name are configured appropriately. A fully qualified LU name is required for an APPC LU.

/luname:" *text*"

Specifies the LU name. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. For a remote APPC LU, the fully qualified LU Name must be unique on the connection, and must not match that of a local LU on that server.

If **/luname:***text* is not specified, *LUalias* is used as the default LU name.

Note
/netname:*text* also has a default (the remote network name configured for the connection supporting this LU). Therefore, the fully qualified LU name (LU name plus network name) can potentially be created by default, if the LU alias and remote control point name are configured appropriately. A fully qualified LU name is required for an APPC LU.

/comment:" *text*"

Adds an optional comment for the LU. The comment can contain as many as 25 characters; enclose the comment in quotes.

/autopartner:{ yes | no }

Specifies whether this LU will automatically be partnered with other APPC LUs.

If automatic partnering is left unspecified, the default is **yes**.

/parallelsess:{ yes | no }

Specifies whether the remote LU supports parallel sessions.

/uninterpname:" *text*"

Specifies the uninterpreted LU name, which is required only when using dependent APPC. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, @ and period (.).

/impmode: *modename*

Designates *modename* as the implicit incoming mode for this LU. A mode must exist before being specified as an implicit incoming mode.

/security:{ none| **hex,***text*| **char,***text* **}**

Configures session security for a remote LU using a cleartext key. The **none** option turns off session-level security. The **hex,***text* option specifies a 16-digit security key in hexadecimal. The **char,***text* option specifies an eight-character security key that can include uppercase and lowercase alphanumeric characters, and the special characters $, @, #, and the period (.).

/addpartner: *LUalias* **,** *mode*

Partners the remote LU with the specified local LU and the specified mode. Both the local LU and the mode must exist before they can be specified as partners. *LUalias* should specify a local LU, not a remote LU; otherwise, an error message is displayed, indicating that no such local LU can be found.

Only one **/addpartner** option can be used in each command.

/delpartner: *LUalias* **,** *mode*

Deletes the pair-listing that includes this remote LU, the specified local LU, and the specified mode. (Does not delete any LUs or modes themselves.)

Only one **/delpartner** option can be used in each command.

/securityex:{ none| **hex,***text*| **char,***text* **}**

Configures session security for a remote LU using a scrambled key. For security purposes, Snacfg displays the security key information in a scrambled format when the /securityex option is specified. To change the security key, use the /security option instead.

See Also

Other Resources

Snacfg Reference

Snacfg Connection

Purpose

Allows you to view, add, delete, or modify connections, including peer connections (necessary for APPC LUs) or downstream connections.

Before configuring a connection, you must configure the server and link service that the connection will use.

Note
Configuration settings specified with snacfg connection correspond to connection settings configured with the SNA Manager .

Syntax

```
          [configpath]  [configpath] connectionname [configpath] connectionname    servernam
  e{  [options]
   [configpath] connectionname [options]
   [configpath] connectionname /
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured connections.

connectionname

Specifies a name for the connection to be configured or viewed. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. A new connection name cannot be the same as any other connection name in the installation, and cannot be the reserved name SNASERVR.

If no options are specified after *connectionname*, the command-line interface displays a list of the configuration settings for the specified connection.

/add

Adds a connection called *connectionname*. To configure the connection, either specify other options after **/add**, or specify configuration options in additional **snacfg connection** commands (using the same *connectionname*).

/delete

Deletes *connectionname*.

Options Used with All Connection Types
/server: *servername*

Specifies the server to which to assign or move the connection. When **/add** is used, this option is required. The server name should be in the format machine_name or \\machine_name\snaservr (for specifying the primary node on the machine) and \\machine_name\snasrv02 (or snasrv03, snasrv04, etc.) for specifying the secondary nodes on the machine.

/conntype:{ 802.2 | SDLC | X.25 | CHANNEL | TWINAX }

Specifies the connection type. When **/add** is used, this option is required.

/comment:" *text*"

Adds an optional comment for the specified connection. The comment can contain as many as 25 characters; enclose the comment in quotes.

/linkservice: *linkname*

Specifies the name of the link service to be used by *connectionname*. The link service type must match the connection type (802.2, SDLC, or X.25), or the **snacfg** command will not run.

In order for the link service to function correctly, it must be installed with the SNA Manager. Link services can also be installed with **snacfg link**; however, the SNA Manager is the recommended interface for installing link services, because it helps ensure that the resulting configuration is functional.

/activation:{ onserverstartup | ondemand | byadministrator }

For outgoing calls on *connectionname*, tells how the connection will be activated: on server startup, on demand, or by the administrator. (For incoming calls, activation is irrelevant, since the connection always begins listening for calls on server startup.)

If no value has been specified for **/activation**, the default for 802.2 (token ring or Ethernet) connections is **onserverstartup**. For all other connections, the default is **ondemand**.

/localblockno: *hexdigits*

Specifies the local block number, a three-digit hexadecimal number. The local block number forms the first part of the Local Node ID, an eight-digit hexadecimal number that identifies the local system.

Do not use 000 or FFF for the local block number. These values are reserved.

For connections to host systems, the local block number should match IDBLK in VTAM.

/localnodeno: *hexdigits*

Specifies the local node number, a five-digit hexadecimal number. The local node number forms the last part of the Local Node ID, an eight-digit hexadecimal number that identifies the local system.

For connections to host systems, the local node number should match IDNUM in VTAM.

/cpname: *text*

Specifies the control point name of the remote node, as it is represented in Format 3 XIDs. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @.

The control point name of the remote node works together with **netname**. If either of these parameters is supplied, the other should also be supplied.

When connecting to a host system and using a remote control point name, the name should match the SSCPNAME parameter in the VTAM Start command for the remote SSCP (the VTAM system).

/netname: *text*

Specifies the name of the network for the remote node, as it is represented in Format 3 XIDs. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @.

The **netname** parameter works together with the control point name of the remote node. If either of these parameters is supplied, the other should also be supplied.

/remoteblockno: *hexdigits*

Specifies the remote block number, a three-digit hexadecimal number. The remote block number forms the first part of the Remote Node ID, an eight-digit hexadecimal number that identifies the remote system.

Do not use 000 or FFF for the remote block number. These values are reserved.

/remoteend:{ Host | Ppeer | downstream | PUPassThrough}

Specifies whether the connection is to be a host, peer, downstream, or passthrough.

/remotenodeno: *hexdigits*

Specifies the remote node number, a five-digit hexadecimal number. The remote node number forms the last part of the Remote Node ID, an eight-digit hexadecimal number that identifies the remote system.

/xidtype:{ format0 | format3 }

Specifies the XID type, the type of identifying information for Host Integration Server to send. The choices are **format0** (Format 0) and **format3** (Format 3). Format 0 sends only the Node ID. Format 3 sends up to 100 bytes of identifying information, including the local node ID and control point name.

If no XID type has been specified, the default is **format3**.

/calldirection:{ Incoming | **Outgoing** | **Both }**

This option specifies the call direction.

/channeladdress: *hex string*

Specifies the channel sub address for channel attach connections. *Hex string* is a two-digit hexadecimal number, valid range 00..FF.

/localsap: *hexnum*

Specifies the local System Access Point (SAP). Enter a hexadecimal number between 04 and EC that is a multiple of 4. For example, *snacfg connection thisconn /localsap:7C*.

/localcpname: *text*

The Local Control Point Name works with the Network Name to identify a system. The maximum length is eight characters.

/localnetname: *text*

The Local Network Name works with the Local Control Point Name to identify a system. The maximum length is eight characters.

/compression:{ None | RLE | LZ9 }

These options offer progressively better compression, but at a progressively higher CPU usage cost.

/passthruconn: *text*

Specifies the name of the PU Passthrough connection.

/peerdlcrole:{ Primary | Secondary | Negotiable }

Specifies the role used in peer-to-peer communications.

/dynamicludef:{ yes | no }

Specifies that this connection supports dynamic remote APPC LU allocation.

Additional Options Used with Downstream Connections
/insert: *luname* **[,*luname*,]**

Assigns a downstream LU or pool to a downstream connection. Separate multiple LU or pool names with commas.

The downstream LU or pool named by ***luname*** must already exist. (A downstream LU can be created with **snacfg LUD**, and a downstream pool can be created with **snacfg poold**.) The connection named by *connectionname* must be a downstream connection. If these conditions are not met, the command is not processed.

/remove: *luname* **[,*luname*,]**

Removes the assignment of a downstream LU or pool to a downstream connection. Separate multiple LU or pool names with commas.

Additional Options Used with 802.2 (Token Ring or Ethernet) Connections
/remotenetaddr: *hexdigits*

Specifies the 12-digit hexadecimal network address of the remote host to which this connection provides access.

If no remote network address has been specified, the default is 400000000000.

/remotesapaddr: *hexdigits*

Specifies the remote SAP address, which is a two-digit hexadecimal number, a multiple of 4, between 04 and EC. A value of 04 is recommended for most installations.

If no remote SAP address has been specified, the default is 04.

/maxbtulen: *value*

Specifies the maximum length for the BTU, which is the number of bytes that can be transmitted in a single data-link control frame.

The range is from 265 through 16393. If no maximum BTU length has been specified, the default is 1929.

/receiveackthresh: *value*

Specifies the receive ACK threshold, the maximum number of frames that the local system can receive from the remote system before sending a response.

The range is from 1 through 127. If no receive ACK threshold has been specified, the default is 2.

/naksendlimit: *value*

Specifies the unacknowledged send limit, the maximum number of frames that the local system can send without receiving a response from the remote system.

The range is from 1 through 127. If no unacknowledged send limit has been specified, the default is 1.

/retrylimit: *value*

Specifies the retry limit, the number of times that the local system should retransmit a frame if no response is received from the remote system.

The range is from 0 through 255. A value of 0 means the system uses its internal default retry limit. If no retry limit has been specified, the default is 10.

/xidretries: *value*

Specifies the XID retries, the number of times that the local system should retransmit an XID (an identifying message) if no response is received from the remote system.

The range is from 0 through 30. If no XID retries value has been specified, the default is 3.

/t1timeout:{ Default| 200ms| 400ms| 600ms| 800ms| 1000ms | 1s| 2s| 3s| 4s | 5s}

Specifies the amount of time that the local system should wait for the remote system to respond to a transmission before the local system tries again.

The values used for **Default** for **t1timeout** are 400 milliseconds for a local ring and 2 seconds for a remote ring.

/t2timeout:{ Default| 40ms| 80ms| 120ms| 160ms| 200ms| 400ms| 800ms| 1200ms| 1600ms| 2000ms}

Select the maximum amount of time that should be allowed before the local system sends an acknowledgment of a received transmission.

The values used for **Default** for **t2timeout** are 80 milliseconds for a local ring and 800 milliseconds for a remote ring.

/titimeout:{ Default| 1s| 2s| 3s| 4s| 5s| 10s| 15s| 20s| 25s}

Select the amount of time that the link can be inactive before the local system treats it as nonfunctioning and shuts it down.

The values used for **Default** for **titimeout** are 5 seconds for a local ring and 25 seconds for a remote ring.

/activatedelay:{ Default | 5s | 10s | 15s | 20s | 25s | 30s | 35s | 40s | 45s | 50s | 55s | 60s | 65s | 70s | 75s | 80s | 85s | 90s | 95s | 100s | 105s | 110s | 115s | 120s | 125s | 130s | 135s | 140s | 145s | 150s | 155s | 160s | 165s | 170s | 175s | 180s | 185s | 190s | 195s | 200s | 205s | 210s | 215s | 220s | 225s | 230s | 235s | 240s | 245s | 250s | 255s }

This value specifies the delay between successive attempts to activate a Host Integration Server connection.

/activateretrylimit:{ None | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 125 | 130 | 135 | 140 | 145 | 150 | 155 | 160 | 165 | 170 | 175 | 180 | 185 | 190 | 195 | 200 | 205 | 210 | 215 | 220 | 225 | 230 | 235 | 240 | 245 | 250 | 255 }

This option specifies the number of times the server will try to activate a connection.

Additional Options Used with SDLC Connections
/dialdata: *value*

Specifies the phone number stored for this connection. For a modem that accepts a phone number from Host Integration Server that is, a modem attached to an SDLC adapter with a built-in serial (COM) port the dial data specifies the telephone number for Host Integration Server to send to the modem. In this case, the number should be in the format expected by the modem. For manually dialed modems, the dial data is displayed in a pop-up message when the connection is started, and can be in any format.

/encoding:{ NRZI | NRZ }

Specifies the encoding scheme to be used by the modem. The choices are **nrzi**, nonreturn to zero inverted, and **NRZ**, nonreturn to zero.

The modem must use the same encoding scheme as the modem at the remote computer. For connections to host systems, the encoding scheme must match the value in the LINE/GROUP definition in VTAM.

If no encoding scheme has been specified, the default is NRZI.

/duplex:{ half | full }

Specifies the modem duplex setting. The choices are **half**, for a half-duplex modem, and **full**, for a full-duplex modem. If you want to use the full-duplex setting, one or more of your adapters must have the constant carrier option set. The constant carrier option is set in the SNA Manager. The constant carrier option can also be set with **snacfg link**; however, the SNA Manager is the recommended interface for setting the constant carrier option and other link service options.

Most Host Integration Server computers will use the default for duplex, **half**.

/datarate:{ high | low }

Specifies the data rate for transmissions between the Host Integration Server communications adapter and the modem. This rate can only be set for certain kinds of modems and adapters; for specific information, see the adapter and modem documentation.

A data rate of **high** gives faster transmissions; **low** gives more reliable transmissions and prevents the transmission errors sometimes caused by poor-quality lines at the high rate.

If no data rate has been specified, the default is **high**.

/polladdress: *hexdigits*

Specifies the poll address, a two-digit hexadecimal number. For connections to a host, the local poll address should match the VTAM PU definition for the ADDR= parameter.

Do not use 00 or FF for the poll address; these values are reserved. If no poll address has been specified, the default is C1.

/idletimeout: *value*

Specifies, in tenths of a second, the idle time-out. The idle time-out is the length of time that the local system should wait for the host to respond to a transmission, before the local system tries again. Too small a time-out can cause connection problems.

The range is from 1 (one-tenth of a second) through 300 (30 seconds). If no idle time-out has been specified, the default is 300 (30 seconds).

/idleretrylimit: *value*

Specifies the idle retry limit, the number of times the local system should try to poll or send data to the host if there is no response.

The range is from 1 through 255. If no idle retry limit has been specified, the default is 10.

/contacttimeout: *value*

Specifies the contact time-out: the length of time, in tenths of a second, which the local system should wait between attempts to make a connection with a remote system.

The range is from 5 (five-tenths of a second) through 300 (30 seconds). If no contact time-out has been specified, the default is 300 (30 seconds).

/contactretrylimit: *value*

Specifies contact retry limit, the maximum number of times the local system should attempt to make a given connection.

The range is from 1 through 20. If no contact retry limit has been specified, the default is 10.

/switchedconntimeout: *value*

Specifies the switched connection establishment time-out; used for switched SDLC lines (standard telephone lines) only. The switched connection establishment time-out is the number of seconds that will be allowed for the user or modem to dial the remote computer's number.

This parameter is ignored by incoming calls.

The range is from 10 through 500 seconds. If no switched connection establishment time-out has been specified, the default is 300.

/multidropprimconn:{ yes | no }

Specifies whether this connection will be a multidrop primary connection.

A multidrop connection is one in which a primary node communicates with multiple secondary nodes concurrently over the same physical transmission medium.

/selectstandby:{ yes | no }

Specifies whether the modem's standby line is set to "on." Standby can only be set for certain kinds of modems; for specific information, see the modem documentation.

If no setting has been specified for standby, the default setting is **no**.

/sdlcmaxbtu: *value*

Specifies the maximum length for the BTU, which is the number of bytes that can be transmitted in a single data-link information frame.

The range is from 265 through 16393. If no maximum BTU length for SDLC has been specified, the default is 265.

/activatedelay:{ Default | 5s | 10s | 15s | 20s | 25s | 30s | 35s | 40s | 45s | 50s | 55s | 60s | 65s | 70s | 75s | 80s | 85s | 90s | 95s | 100s | 105s | 110s | 115s | 120s | 125s | 130s | 135s | 140s | 145s | 150s | 155s | 160s | 165s | 170s | 175s | 180s | 185s | 190s | 195s | 200s | 205s | 210s | 215s | 220s | 225s | 230s | 235s | 240s | 245s | 250s | 255s }

This value specifies the delay between successive attempts to activate a Host Integration Server connection.

/activateretrylimit:{ None | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 125 | 130 | 135 | 140 | 145 | 150 | 155 | 160 | 165 | 170 | 175 | 180 | 185 | 190 | 195 | 200 | 205 | 210 | 215 | 220 | 225 | 230 | 235 | 240 | 245 | 250 | 255 }

This option specifies the number of times the server will try to activate a connection.

Additional Options Used with SDLC Peer Connections
/multidropprimconn:{ yes | no }

Specifies whether this server is to be the primary station for a multidrop connection on a leased SDLC line.

/pollrate: *value*

Specifies the poll rate in polls per second.

The range is from 1 through 50. If no poll rate has been specified, the default is 5.

/activatedelay:{ Default | 5s | 10s | 15s | 20s | 25s | 30s | 35s | 40s | 45s | 50s | 55s | 60s | 65s | 70s | 75s | 80s | 85s | 90s | 95s | 100s | 105s | 110s | 115s | 120s | 125s | 130s | 135s | 140s | 145s | 150s | 155s | 160s | 165s | 170s | 175s | 180s | 185s | 190s | 195s | 200s | 205s | 210s | 215s | 220s | 225s | 230s | 235s | 240s | 245s | 250s | 255s }

This value specifies the delay between successive attempts to activate a Host Integration Server

/activateretrylimit:{ None | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 125 | 130 | 135 | 140 | 145 | 150 | 155 | 160 | 165 | 170 | 175 | 180 | 185 | 190 | 195 | 200 | 205 | 210 | 215 | 220 | 225 | 230 | 235 | 240 | 245 | 250 | 255 }

This option specifies the number of times the server will try to activate a connection.

Additional Options Used with X.25 Connections
/remotex25addr: *hexdigits*

Specifies the remote X.25 address, which identifies the remote system on an X.25 network. The address usually consists of 12 hexadecimal digits, but can contain up to 15 hexadecimal digits.

/x25maxbtu: *value*

Specifies the maximum length for the BTU, which is the number of bytes that can be transmitted in a single data-link information frame.

The range is from 265 through 16393. If no maximum BTU length for X.25 has been specified, the default (for host connections) is 265.

/virtualcircuit:{ perm | switched }

Specifies the type of virtual circuit used by the connection. The choices are **perm** and **switched**. A **perm** circuit is constantly active and uses a preset destination address. A **switched** circuit is called and cleared dynamically, and uses a destination address that is supplied when the circuit is called.

If no virtual circuit type has been specified, the default is **switched**.

/pvcalias: *value*(for PVC only)

Specifies the PVC alias, the number that identifies the PVC channel: 1 for the first channel, 2 for the second, and so on. Used for PVCs only.

The range is from 1 through the number of configured PVC channels. If no PVC alias has been specified, the default is 1.

/packetsize: *value*(for PVC only)

Specifies packet size, the maximum number of data bytes (not header bytes) to be sent in a frame on this X.25 network. Used for PVCs only.

The possible values are 64, 128, 256, 512, and 1024. If no packet size has been specified, the default is 128.

/windowsize: *value*(for PVC only)

Specifies window size, the maximum number of frames that the local system can send without receiving a response from the remote system, on this X.25 network. Used for PVCs only.

The range is from 1 through 7. If no window size has been specified, the default is 2.

/facilitydata: *text*(for SVC only)

Specifies the codes for any facility data required by the network provider or by the administrator of the remote system. Used for SVCs only. Facility data can include as many as 126 hexadecimal characters (63 hexadecimal bytes).

Facility data is a coded string of information often used to request nondefault functions from the X.25 network for a particular SVC connection.

/userdata: *text*(for SVC only)

Specifies the codes for any user data required by the network provider. Used for SVCs only. The user data must be an even number of hexadecimal characters, up to the maximum of 32 characters.

User data is a coded string of information, specifying items such as the communications protocol used by the X.25 network (for SNA, this protocol must be QLLC, specified by C3).

The default for user data is C3; this specifies the QLLC protocol.

/activatedelay:{ Default | **5s** | **10s** | **15s** | **20s** | **25s** | **30s** | **35s** | **40s** | **45s** | **50s** | **55s** | **60s** | **65s** | **70s** | **75s** | **80s** | **85s** | **90s** | **95s** | **100s** | **105s** | **110s** | **115s** | **120s** | **125s** | **130s** | **135s** | **140s** | **145s** | **150s** | **155s** | **160s** | **165s** | **170s** | **175s** | **180s** | **185s** | **190s** | **195s** | **200s** | **205s** | **210s** | **215s** | **220s** | **225s** | **230s** | **235s** | **240s** | **245s** | **250s** | **255s }**

This value specifies the delay between successive attempts to activate a Host Integration Server connection.

/activateretrylimit:{ None | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **45** | **50** | **55** | **60** | **65** | **70** | **75** | **80** | **85** | **90** | **95** | **100** | **105** | **110** | **115** | **120** | **125** | **130** | **135** | **140** | **145** | **150** | **155** | **160** | **165** | **170** | **175** | **180** | **185** | **190** | **195** | **200** | **205** | **210** | **215** | **220** | **225** | **230** | **235** | **240** | **245** | **250** | **255 }**

This option specifies the number of times the server will try to activate a connection.

Additional Options Used with Channel Connections
/activatedelay:{ Default | **5s** | **10s** | **15s** | **20s** | **25s** | **30s** | **35s** | **40s** | **45s** | **50s** | **55s** | **60s** | **65s** | **70s** | **75s** | **80s** | **85s** | **90s** | **95s** | **100s** | **105s** | **110s** | **115s** | **120s** | **125s** | **130s** | **135s** | **140s** | **145s** | **150s** | **155s** | **160s** | **165s** | **170s** | **175s** | **180s** | **185s** | **190s** | **195s** | **200s** | **205s** | **210s** | **215s** | **220s** | **225s** | **230s** | **235s** | **240s** | **245s** | **250s** | **255s }**

This value specifies the delay between successive attempts to activate a Host Integration Server connection.

/activateretrylimit:{ None | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **45** | **50** | **55** | **60** | **65** | **70** | **75** | **80** | **85** | **90** | **95** | **100** | **105** | **110** | **115** | **120** | **125** | **130** | **135** | **140** | **145** | **150** | **155** | **160** | **165** | **170** | **175** | **180** | **185** | **190** | **195** | **200** | **205** | **210** | **215** | **220** | **225** | **230** | **235** | **240** | **245** | **250** | **255 }**

This option specifies the number of times the server will try to activate a connection.

/controlunit: *value (0x0-0xf)*

Sets the value of the control unit image number.

See Also
Other Resources
Snacfg Reference

Snacfg CPIC

Purpose

Allows you add, delete, modify, or view a CPI-C symbolic destination name. Also allows you to view the command that would create a specified CPI-C symbolic destination name.

Note
Settings specified with snacfg cpic correspond to CPI-C symbolic destination names configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] cpicname [configpath] cpicname
     { text  |  hexstring }
     { btexttext  |  text }
     text{  |  |  }
     [text] [text] [text]
   [configpath] cpicname []
   [configpath] cpicname [configpath] cpicname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured symbolic destination names.

cpicname

Specifies the symbolic destination name on which to carry out actions. A symbolic destination name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @.

If no options are specified after *cpicname*, the configuration settings for the specified symbolic destination name are displayed.

/add

Adds a symbolic destination name called *cpicname*. When you use the **/add** option, you must include the other options shown in the preceding syntax.

/delete

Deletes *cpicname*.

/print

Causes the display of the **snacfg** command that would create the specified symbolic destination name. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. See the information about command files earlier in this section.

Options for CPI-C Commands

/comment:" *text*"

Adds an optional comment for the symbolic destination name. The comment can contain as many as 25 characters; enclose the comment in quotes.

Partner TP options(use one or the other, but not both):

/appltpname:"**text**"

Specifies that the partner TP is an application TP, and provides the name. The name can be from 1 through 64 characters

long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase.

If you specify both an application TP name and a service TP name in the same command, the command is rejected. If you specify an application TP for an existing symbolic destination name, it overrides any previous TP name (whether application TP or service TP).

/svcetpname:**hexstring**

Specifies that the partner TP is a service TP, and provides the hexadecimal string identifying the TP. The string can be from one through eight hexadecimal digits long.

If you specify both a service TP name and an application TP name in the same command, the command is rejected. If you specify a service TP for an existing symbolic destination name, it overrides any previous TP name (whether application TP or service TP).

Partner LU options(use one or the other, but not both):

/netname:"**text**" /luname:"**text**"

Identifies the partner LU by fully qualified LU name (network name plus LU name). Each part of the fully qualified name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase.

If you specify both a fully qualified LU name and an LU alias in the same command, the command is rejected. If you specify a fully qualified LU name for an existing symbolic destination name, it overrides any previous LU setting (whether name or alias).

/lualias:"**text**"

Identifies the partner LU by LU alias. The alias can be from one through eight characters long, and can contain alphanumeric characters and the special characters %, $, #, and @. Lowercase letters are converted to uppercase.

If you specify both an LU alias and a fully qualified LU name in the same command, the command is rejected. If you specify an LU alias for an existing symbolic destination name, it overrides any previous LU setting (whether name or alias).

/modename:" *text*"

Specifies the mode. The mode must already exist.

/seckeytype:{ none| **same**| **program}**

Sets the conversation security type. If **program** is specified, one or both of the following options can be set:

/secuserid:" **text**"

Specifies the user ID to use when the security type is **program**. The ID can contain from 1 through 10 characters.

/secpassword:" **text**"

This parameter is optional; it specifies the password to use with the user ID. The password can contain from 1 through 10 characters. A password need not be specified even if the security type is **program** and the user ID is set. If a display is created showing the symbolic destination name, the password will not be displayed.

See Also
Other Resources
Snacfg Reference

Snacfg Diagnostic

Purpose

Allows you to view and configure settings for event logging (auditing), and view or change the connection designated to receive NetView alerts.

Note
Configuration settings specified with snacfg diagnostic correspond to diagnostic settings configured with the SNA Manager.

Syntax

```
snacfg [#configpath] diagnostic /list
snacfg [#configpath] diagnostic [options]
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list that tells the settings for event logs (auditing), the name of the NetView management connection (if one has been specified), and the default connection for the **DISPLAY** verb (if one has been specified).

/logserver: *servername*

Specifies a centralized server on which event logs for this Host Integration Server installation should be stored.

If **/logserver:** is typed without *servername*, the setting reverts to the default, which is to store event logs for the local server on the local server.

/auditlevel:{ 6| 8| 10| off}

Sets the level of Host Integration Server events to be recorded in the Windows Event Log. These events can be viewed with the Windows Event Viewer, which is described in the Windows documentation. The following table describes the available levels:

Level	Description	Events to be recorded
6	Detailed problem analysis	All events that can be recorded
8	General information messages	General activity but not all events
10	Significant system events	Major events only
off	Auditing disabled	No events

/popupserver: *servername*

Specifies the server to which pop-up error messages for this Host Integration Server installation should be routed.

Pop-up messages will always appear on the local server; routing them to a remote server means they will appear on both the remote and local server.

If **/popupserver:** is typed without *servername*, the setting reverts to the default, which is the local server.

/netviewconn: *connectionname*

Specifies the name of the connection through which NetView alerts are sent. The connection called *connectionname* must already exist.

If **/netviewconn:** is typed without *connectionname*, the setting is cleared so that the configuration contains no NetView connection name.

/cnosdisplayconn: *connectionname*

Specifies the default connection to be used for the **DISPLAY** verb when no connection is provided. The connection called *connectionname* must already exist.

If **/cnosdisplayconn:** is typed without *connectionname*, the setting is cleared so that the configuration contains no default connection for the **DISPLAY** verb.

See Also

Other Resources

Snacfg Reference

Snacfg LINK

Purpose

Allows you to view, delete, or modify adapters.

Note
This command has been superseded by **Linkcfg**, as described in Linkcfg.

Important
If you specify adapter properties with **snacfg link**, these properties must match existing properties in your installation, or a n onfunctioning configuration may result. To protect against errors with **snacfg link**, use the SNA Manager instead.

Before configuring link services, you must use the SNA Manager to add the server on which the link services will be located.

Syntax

```
        [configpath]
```

Recommended Syntax

```
        [configpath]servernamelinkname [configpath]servernamelinkname [options]
[configpath]servernamelinkname [options]
[configpath]servernamelinkname
```

Other Available Syntax

```
        [configpath]linkname [configpath]linkname [options]
[configpath]linkname [options]
[configpath]linkname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured link services.

servername

Specifies the name of the server on which to view or change link services. Separate the name of the server from the name of the link service with a colon (**:**). It is recommended that *servername* be included in every **snacfg link** command that includes *linkname*. Without *servername*, if there is more than one link service called *linkname* in the installation, it is difficult to predict which link service called *linkname* will be affected by the command. The **snacfg link** command does not necessarily default to the local server if *servername* is omitted.

linkname

Specifies the name of the link service to view, modify, or delete. Link services names can contain from one through eight alphanumeric characters.

For adding link services, the recommended method is to use the SNA Manager, not **snacfg link**.

If no options are specified after *linkname*, the result is a list of the configuration settings for the specified link.

/add

Adds a link service called *linkname*. To configure the link service, either specify other options after **/add**, or specify configuration options in additional **snacfg link** commands (using the same *servername***:***linkname* combination).

For adding link services, the recommended method is to use the SNA Manager, not **snacfg link**.

/delete

Deletes *linkname*.

Options for Link Services
/server: *servername*

Specifies the server on which to install the link service.

/linktype:{ Token| **Ether**| **SDLC**| **X25**| **Channel | Twinax }**

Specifies the type of adapter with which *linkname* works.

/linetype:{ leased| **softdial**| **manual}**(for SDLC lines only)

Specifies the type of SDLC line (and, where applicable, the modem) that the link service will use:

- A **leased** line is a telecommunications line committed solely to SDLC communications with a particular remote system.
- A **softdial** line is a switched SDLC line on which the modem is dialed automatically by Host Integration Server. Such a modem must be attached to an SDLC adapter with a built-in serial (COM) port. Otherwise it is considered a **manual** modem.
- A **manual** line is a switched SDLC line on which the modem stores a phone number, or on which the modem is dialed manually.

/carrier:{ on| **off}**(for SDLC lines only)

Specifies whether the constant carrier option for an SDLC line is **on** or **off**.

See Also
Other Resources
Snacfg Reference

Snacfg LU

Purpose

Allows you to view, add, delete, or modify 3270 LUs, on three types of connections: 802.2, SDLC, and/or X.25. Also allows you to assign 3270 LUs to LU pools that have already been configured.

Before configuring an LU, you must configure the connection that the LU will use. Also, before assigning an LU to an LU pool, you must create the LU and the LU pool.

Note
Configuration settings specified with snacfg lu correspond to 3270 LU settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] luname [configpath] lunameconnectionnamevalue [options]
  [configpath] luname [options]
  [configpath] luname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured 3270 LUs.

luname

Specifies the name of the 3270 LU to view, add, modify, or delete. A 3270 LU name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. A 3270 LU name cannot be the same as any other LU name or pool name (except for APPC LU names) on the server.

If no options are specified after *luname,* the result is a list of the configuration settings for the specified LU.

/add

Adds a 3270 LU called *luname*. To configure the 3270 LU, either specify other options after **/add**, or specify configuration options in additional **snacfg lu** commands (using the same *luname*).

/delete

Deletes *luname*.

Options for 3270 LUs
/connection: *connectionname*

Specifies the connection to which the 3270 LU should be assigned or moved. When **/add** is used, this option is required.

/lunumber: *value*

Specifies the LU number, which identifies the LU on its connection. When /**add** is used, this option is required. Check with the administrator of the host system for the correct value; it should match the LOCADDR= parameter of the LU definition in VTAM or the NCP Gen on the host.

If the number that you specify has already been assigned to an LU or an APPC LU-LU pair on the intended connection, the command fails.

The range is from 1 through 254.

/pool: *poolname*

Assigns or moves the 3270 LU to *poolname*. A 3270 LU can be assigned to only one pool.

The pool specified with *poolname* must already exist. A 3270 LU pool can be created with the **snacfg pool** command.

If **/pool:** is typed without *poolname*, the specified LU is removed from whichever pool it is assigned to.

/lutype:{ display | printer }

Specifies whether the 3270 LU will be used for **display** (terminal emulation) or a **printer** (that is, a local printer or a local-area network printer, attached to a PC).

If no LU type has been specified, the default is **display**.

/displaymodel:{ mod2 | mod3 | mod4 | mod5 | 2 | 3 | 4 | 5 }

Specifies the display model; applies only when the **lutype** is **display**. The following display types are available:

- Model 2 is 24 lines by 80 characters.
- Model 3 is 32 lines by 80 characters.
- Model 4 is 43 lines by 80 characters.
- Model 5 is 27 lines by 132 characters.

Some emulators can only emulate certain display models. For more information, see your emulator documentation.

When a 3270 display LU is assigned to a 3270 LU pool, the display model setting of the pool overrides the setting of the LU.

If no display model has been specified, the default is Model 2.

/allowmodeloverride:{ yes | no }

Specifies whether the user is allowed to override the display model type by using the 3270 terminal emulation program.

When a 3270 display LU is assigned to a 3270 LU pool, the model override setting of the pool overwrites the setting of the LU.

If no setting has been specified for this parameter, the default is **no**.

/associate:" *text*"

This allows you to associate a specific 3270 printer LU with a 3270 display LU. Any user or group with permission to use the display LU will also have access to the associate printer LU. The text is the printer LU name.

/unassociate:" *text*"

This allows you to unassociate a 3270 printer LU from a 3270 display LU. It is not necessary to enter any text. **Snacfg** will ignore the text and unassociate the LU. To unassociate the printer LU, use the display LU name.

/comment:" *text*"

Adds an optional comment for the specified LU. The comment can contain as many as 25 characters; enclose the comment in quotes.

See Also
Other Resources
Snacfg Reference

Snacfg LUA

Purpose

Allows you to view, add, delete, or modify LUA LUs on three types of connections: 802.2, SDLC, and/or X.25. Also allows you to assign LUA LUs to LU pools that have already been configured.

Before configuring an LU, you must configure the connection that the LU will use. Also, before assigning an LU to an LU pool, you must create the LU and the LU pool.

Note
Configuration settings specified with snacfg lua correspond to LUA LU settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] luname [configpath] lunameconnectionnamevalue [options]
  [configpath] luname [options]
  [configpath] luname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured LUA LUs.

luname

Specifies the name of the LUA LU to view, add, modify, or delete. An LUA LU name can be from one through eight characters long and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. An LUA LU name cannot be the same as any other LU name or pool name (except for APPC LU names) on the server.

If no options are specified after *luname*, the result is a list of the configuration settings for the specified LU.

/add

Adds an LUA LU called *luname*. To configure the LUA LU, either specify other options after **/add**, or specify configuration options in additional **snacfg lua** commands (using the same *luname*).

/delete

Deletes *luname*.

Options for LUA LUs

/connection: *connectionname*

Specifies the connection to which the LUA LU should be assigned or moved. When **/add** is used, this option is required.

/lunumber: *value*

Specifies the LU number, which identifies the LU on its connection. When **/add** is used, this option is required. Check with the administrator of the host system for the correct value; it should match the LOCADDR= parameter of the LU definition in VTAM or the NCP Gen on the host.

If the number that you specify has already been assigned to an LU or an APPC LU-LU pair on the intended connection, the command fails.

The range is from 1 through 254.

/pool: *poolname*

Assigns or moves the LUA LU to *poolname*. An LUA LU can be assigned to only one pool. The pool specified with *poolname* must already exist. An LUA LU pool can be created with the **snacfg poola** command.

If **/pool:** is typed without *poolname*, the specified LU is removed from whichever pool it is assigned to.

/highpriority:{ yes | no }

Specifies whether this LU will be given priority over low-priority LUs.

When an LUA LU is assigned to an LUA LU pool, the priority setting of the pool overwrites the priority setting of the LU.

/comment:" *text*"

Adds an optional comment for the specified LU. The comment can contain as many as 25 characters; enclose the comment in quotes.

See Also
Other Resources
Snacfg Reference

Snacfg LUD

Purpose

Allows you to view, add, delete, or modify downstream LUs. Also allows you to assign downstream LUs to LU pools that have already been created, and allows you to view the command that would create a specified downstream LU.

Note
Configuration settings specified with **snacfg lud** correspond to downstream LU settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] luname [configpath] lunameconnectionnamevalue [options]
  [configpath] luname [options]
  [configpath] luname [configpath] luname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured downstream LUs.

luname

Specifies the name of the downstream LU to view, add, modify, or delete. A downstream LU name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The name cannot be the same as any other LU or pool name that uses this connection.

If no options are specified after *luname*, the result is a list of the configuration settings for the specified LU.

/add

Adds a downstream LU called *luname*. To configure the downstream LU, either specify other options after **/add**, or specify configuration options in additional **snacfg lud** commands (using the same *luname*).

/delete

Deletes *luname*.

/print

Causes the display of the **snacfg** command that would create the specified downstream LU. The displayed command does not contain the word **snacfg**, so it can be redirected to a command file. See the information about command files earlier in this section.

Options for Downstream LUs
/connection:connectionname

Specifies the connection to which the downstream LU should be assigned or moved. When **/add** is used, this option is required.

/lunumber: *value*

Specifies the LU number, which identifies the LU on its connection. When **/add** is used, this option is required. Check with the administrator of the host system for the correct value; it should match the LOCADDR= parameter of the LU definition in VTAM or in the NCP Gen.

The LU number identifies the LU to the host system.

If the number you specify has already been assigned to an LU or an APPC LU-LU pair on the intended connection, the

command fails.

The range is from 1 through 254.

/pool: *poolname*

Assigns or moves the downstream LU to *poolname*. A downstream LU can be assigned to only one pool. The pool specified with *poolname* must already exist. A downstream LU pool can be created with the **snacfg poold** command.

If **/pool:** is typed without *poolname*, the specified LU is removed from whichever pool it is assigned to.

/comment:" *text*"

Adds an optional comment for the specified LU. The comment can contain as many as 25 characters; enclose the comment in quotes.

See Also

Other Resources

Snacfg Reference

Snacfg Mode

Purpose

Lets you add, delete, modify, or view an APPC mode. Also lets you to view the command that would create a specified mode.

Note
Configuration settings specified with snacfg mode correspond to APPC mode settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] modename [configpath] modename [options]
  [configpath] modename [options]
  [configpath] modename [configpath] modename
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured modes.

modename

Specifies the name of the mode on which to carry out actions. A mode name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The mode name cannot be the same as any other mode name in the subdomain of the server.

If no options are specified after *modename*, the configuration settings for the specified mode are displayed.

/add

Adds a mode called *modename*. To configure the mode, either specify other options after **/add**, or specify configuration options in additional **snacfg mode** commands (using the same *modename*).

/delete

Deletes *modename*.

/print

Causes the display of the **snacfg** command that would create the specified mode. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. See the information about command files earlier in this section.

Options for APPC Modes
/comment:" *text*"

Adds an optional comment for the mode. The comment can contain as many as 25 characters; enclose the comment in quotes.

/sessionlim: *value*

Specifies the parallel session limit.

The range is from 1 through 254. If no parallel session limit has been specified, the default is 1.

/conwin: *value*

Specifies the minimum contention winner limit.

The range is from 0 through the parallel session limit. If no minimum contention winner limit has been specified, the default

is 0.

/conlose: *value*

Specifies the partner minimum contention winner limit.

The range is from 0 through the parallel session limit. If no partner minimum contention winner limit has been specified, the default is 0.

/autoact: *value*

Specifies the automatic activation limit.

The range is from 0 through the minimum contention winner limit.

/autopartner:{ yes | no }

Specifies whether this mode will be used in automatic partnering of APPC LUs.

If automatic partnering is left unspecified, the default is **yes**.

/highpriority:{ yes | no }

Specifies whether communication with this mode will be given preference over low-priority communication.

If the high-priority setting is left unspecified, the default is **yes**.

/pacesendcnt: *value*

Specifies the pacing send count. A value of 0 represents an unlimited number of frames.

The range is from 0 through 63. If no pacing send count has been specified, the default is 4.

/pacerecvcnt: *value*

Specifies the pacing receive count. A value of 0 represents an unlimited number of frames.

The range is from 0 through 63. If no pacing receive count has been specified, the default is 4.

/maxsendru: *value*

Specifies the maximum size for RUs sent by the TP(s) on the local system.

The range is from 256 through 16384. If no maximum send RU size has been specified, the default is 1024.

/maxrecvru: *value*

Specifies the maximum size for RUs received from the TP(s) on the remote system.

The range is from 256 through 16384. If no maximum receive RU size has been specified, the default is 1024.

/maxsendcomp:{ None | RLE | LZ9}

These options offer progressively better compression, but at a progressively higher CPU usage cost.

/maxrecvcomp:{ None | RLE | LZ9}

These options offer progressively better compression, but at a progressively higher CPU usage cost.

/allowcomp:{ yes | no}

If LZ9 is used, this option controls whether data is compressed using RLE before being further compressed using LZ9.

/endpointcomp:{ yes | no}

This option controls whether intermediate nodes may use compression if one of the endpoints does not support compression or does not want to use it.

See Also

Other Resources

Snacfg Reference

Snacfg Pool

Purpose

Allows you to view, add, delete, or modify 3270 LU pools.

To assign existing 3270 LUs to a 3270 LU pool, first configure the pool with the **snacfg pool** command (including options), then add the LUs with the **snacfg lu** command (using the **/pool:***poolname* option).

Note
Configuration settings specified with snacfg pool correspond to 3270 LU pool settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] poolname [configpath] poolname [options]
  [configpath] poolname [options]
  [configpath] poolname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured 3270 LU pools.

poolname

Specifies the name of the 3270 LU pool to view, add, modify, or delete. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The name cannot be the same as any other pool name or LU name (other than APPC LU names) in the installation.

If no options are specified after *poolname*, the result is a list of the configuration settings for the specified pool.

/add

Adds a 3270 LU pool called *poolname*. To configure the 3270 LU pool, either specify other options after **/add**, or specify configuration options in additional **snacfg pool** commands (using the same *poolname*).

/delete

Deletes *poolname*.

Options for 3270 LU Pools

/displaymodel:{ mod2 | mod3 | mod4 | mod5 | 2 | 3 | 4 | 5 }

Specifies the model number of the LUs that will be added to this pool. (Only display LUs can be pooled; printer LUs cannot be pooled.) The following display models are available:

- Model 2 is 24 lines by 80 characters.
- Model 3 is 32 lines by 80 characters.
- Model 4 is 43 lines by 80 characters.
- Model 5 is 27 lines by 132 characters.

Some emulators can only emulate certain display models. For more information, see your emulator documentation.

The display model setting of a pool overwrites the setting of any 3270 LU assigned to the pool.

If no display model has been specified for a pool, the default is Model 2.

/allowmodeloverride:{ yes | no }

Specifies whether the user is allowed to override the display model type of the LU by using the 3270 terminal emulation program.

The model override setting of a pool overwrites the setting of any 3270 LU assigned to the pool.

If no setting has been specified for this parameter, the default is **no**.

/comment:" *text*"

Adds an optional comment to the specified 3270 LU pool. The comment can contain as many as 25 characters; enclose the comment in quotes.

/assocprint:{ yes | no }

Specifies that the LU pool contains display LUs with associated 3270 printers.

See Also
Other Resources
Snacfg Reference

Snacfg PoolA

Purpose

Allows you to view, add, delete, or modify LUA LU pools.

To assign existing LUA LUs to an LUA LU pool, first configure the pool with the **snacfg poola** command (including options), then add the LUs with the **snacfg lua** command (using the **/pool:***poolname* option).

Note
Configuration settings specified with snacfg poola correspond to LUA LU pool settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] poolname [configpath] poolname [options]
  [configpath] poolname [options]
  [configpath] poolname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured LUA LU pools.

poolname

Specifies the name of the LUA LU pool to view, add, modify, or delete. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The name cannot be the same as any other pool name or LU name (other than APPC LU names) in the installation.

If no options are specified after *poolname*, the result is a list of the configuration settings for the specified pool.

/add

Adds an LUA LU pool called *poolname*. To configure the LUA LU pool, either specify other options after **/add**, or specify configuration options in additional **snacfg poola** commands (using the same *poolname*).

/delete

Deletes *poolname*.

Options for LUA LU Pools

/highpriority:{ yes | no }

Specifies whether LUs in this pool will be given priority over low-priority LUs. The priority setting of a pool overwrites the setting of any LUA LU assigned to the pool.

/comment:" *text*"

Adds an optional comment to the specified LUA LU pool. The comment can contain as many as 25 characters; enclose the comment in quotes.

See Also

Other Resources

Snacfg Reference

Snacfg PoolD

Purpose

Allows you to view, add, or delete downstream LU pools. Also allows you to view the command that would create a specified downstream LU pool.

To assign existing downstream LUs to a downstream LU pool, first configure the pool with the **snacfg poold** command; then add the LUs with the **snacfg lud** command (using the **/pool:***poolname* option).

Syntax

```
          [configpath]  [configpath] poolname [configpath] poolname [] [text]
   [configpath] poolname [configpath] poolname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured downstream LU pools.

poolname

Specifies the name of the downstream LU pool to view, add, or delete. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The name cannot be the same as any other pool name or LU name (other than APPC LU names) in the subdomain.

If no options are specified after *poolname*, the result is a list of the configuration settings for the specified pool.

/add

Adds a downstream LU pool called *poolname*.

/delete

Deletes *poolname*.

/print

Causes the display of the **snacfg** command that would create the specified downstream LU pool. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. See the information about command files earlier in this section.

/comment:" *text*"

Adds an optional comment to the specified downstream LU pool. The comment can contain as many as 25 characters; enclose the comment in quotes.

See Also

Other Resources

Snacfg Reference

Snacfg PrintServer

Purpose

Allows you to add, delete, or view printer servers from one subdomain or multiple subdomains.

Note
Configuration settings specified with snacfg printserver correspond to settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] servername [configpath] servername
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of print servers in the Host Integration Server subdomain.

/add

Adds the print server name of the Host Integration Server computer that is running the Host Integration Server Host Print Service. To configure the server name, specify the name after the **/add**. The print server name must match the computer name of the computer running Host Print Service.

You can add print servers for Host Integration Server computers that do not yet exist. For example, you can prepare a configuration file to be used at another location and add the server names using this parameter. Until the computers are configured with Host Integration Server running Host Print Service, SNA Manager will show them as offline.

/delete

Deletes the print server.

See Also

Other Resources

Snacfg Reference

Snacfg PrintSession3270

Purpose

Allows you to add, delete, modify, or view 3270 print sessions defined in the Host Print Service.

Note
Configuration settings specified with snacfg printsession3270 correspond to local print server settings configured with the SNA Manager.

Note
The /feedignorefinal option is no longer supported. By default, this value is enabled. A PDF file can be used to control this behavior if required.

Syntax

```
          [configpath]  [configpath]  [configpath] printsession3270nameservernameLUname[opti
  ons]
   [configpath] printsession3270name [options]
   [configpath] printsession3270name
```

where

***#** configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of 3270 print sessions.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a print session to the Host Print Service. To configure the print session, you must specify the server name and the configured 3270 printer LU name after the **/add** using the **/server:***servername* and **/luname:***Luname* options.

/delete

Deletes the printer session. To delete the print session, you must specify the server name and the configured 3270 printer LU name after the **/delete**.

Options for 3270 Print Sessions
/autoactivate:{ yes | no }

Specifies whether the printer session will automatically activate when Host Integration Server is started. The default is yes.

/bestfit:{ yes | no }

Specifies whether to scale the output to the paper size. The default is **yes**.

/codepage: {Country | Custom}

This defines the host code page language in which the print jobs are output. The default is **Country** and the default language is **English (United States) [037]**. To change the default language, provide the number of the host code page of the country/region you want with the **/country** option.

If you want to use a custom file for the host code page, you must use **/customfile:***text,* where the *text* value is the name of

the file containing the specifications for the print job.

Host Code Page Numbers and Corresponding Language

Host Code Page Number	Language
0	"Afrikaans [500]"
1	"Albanian [870]"
2	"Arabic (Algeria) [420]"
3	"Arabic (Kingdom of Bahrain) [420]"
4	"Arabic (Egypt) [420]"
5	"Arabic (Iraq) [420]"
6	"Arabic (Jordan) [420]"
7	"Arabic (Kuwait) [420]"
8	"Arabic (Lebanon) [420]"
9	"Arabic (Libya) [420]"
10	"Arabic (Morocco) [420]"
11	"Arabic (Oman) [420]"
12	"Arabic (Qatar) [420]"
13	"Arabic (Saudi Arabia) [420]"
14	"Arabic (Syria) [420]"
15	"Arabic (Tunisia) [420]"
16	"Arabic (U.A.E.) [420]"
17	"Arabic (Yemen) [420]"
18	"Basque [284]"
19	"Belarusian [1025]"
20	"Bulgarian [1025]"
21	"Catalan [284]"
22	"Chinese (PRC) [935]"
23	"Chinese (Singapore) [935]"
24	"Chinese (Hong Kong SAR) [937]"

25	"Chinese (Macao SAR) [937]"
26	"Chinese (Taiwan) [937]"
27	"Croatian [870]"
28	"Czech [870]"
29	"Danish [277]"
30	"Dutch (Belgium) [500]"
31	"Dutch (Standard) [037]"
32	"English (Australian) [037]"
33	"English (Belize) [500]"
34	"English (Canadian) [037]"
35	"English (Caribbean) [500]"
36	"English (Ireland) [285]"
37	"English (Jamaica) [500]"
38	"English (New Zealand) [037]"
39	"English (South Africa) [037]"
40	"English (Trinidad) [500]"
41	"English (United Kingdom) [285]"
42	"English (United States) [037]"
43	"Estonian [1112]"
44	"Faeroese [277]"
45	"Finnish [278]"
46	"French (Belgium) [500]"
47	"French (Canadian) [037]"
48	"French (Luxembourg) [500]"
49	"French (Standard) [297]"
50	"French (Swiss) [500]"
51	"German (Austrian) [273]"

52	"German (Liechtenstein) [500]"
53	"German (Luxembourg) [500]"
54	"German (Standard) [273]"
55	"German (Swiss) [500]"
56	"Greek [423]"
57	"Greek (Modern) [875]"
58	"Hebrew [424]"
59	"Hungarian [870]"
60	"Icelandic [871]"
61	"Indonesian [037]"
62	"Italian [280]"
63	"Italian (Swiss) [500]"
64	"International [500]"
65	"Japanese (Extend Katakana) [930]"
66	"Japanese (English-lower) [931]"
67	"Japanese (Extend English) [939]"
68	"Japanese (Katakana) [290]"
69	"Korean [933]"
70	"Latvian [1112]"
71	"Lithuanian [1112]"
72	"Macedonian [1025]"
73	"Malay [037]"
74	"Norwegian (Bokmal) [277]"
75	"Norwegian (Nynorsk) [277]"
76	"Polish [870]"
77	"Portuguese (Brazil) [037]"
78	"Portuguese (Portugal) [037]"

79	"Romanian [870]"
80	"Russian [880]"
81	"Serbian (Cyrillic) [1025]"
82	"Serbian (Latin) [870]"
83	"Slovak [870]"
84	"Slovenian [870]"
85	"Spanish (Argentina) [284]"
86	"Spanish (Bolivia) [284]"
87	"Spanish (Chile) [284]"
88	"Spanish (Columbia) [284]"
89	"Spanish (Costa Rica) [284]"
90	"Spanish (Dominican Rep.) [284]"
91	"Spanish (Ecuador) [284]"
92	"Spanish (El Salvador) [284]"
93	"Spanish (Guatemala) [284]"
94	"Spanish (Honduras) [284]"
95	"Spanish (Mexico) [284]"
96	"Spanish (Modern Sort) [284]"
97	"Spanish (Nicaragua) [284]"
98	"Spanish (Panama) [284]"
99	"Spanish (Paraguay) [284]"
100	"Spanish (Peru) [284]"
101	"Spanish (Puerto Rico) [284]"
102	"Spanish (Trad. Sort) [284]"
103	"Spanish (Uruguay) [284]"
104	"Spanish (Venezuela) [284]"

105	"Swedish [278]"
106	"Thai [838]"
107	"Turkish [905]"
108	"Turkish (Latin-5) [1026]"
109	"Ukrainian [1025]"

/collate:{ yes | no }

Adds an option to collate pages sequentially.

/color:{ yes | no }

For color printers, printing will be gray scale if nothing is selected.

/comment:" *text*"

Adds an optional comment for the printer session. It can contain as many as 25 characters; enclose the comment in quotes.

/copies: *value*

Specifies the number of copies for printing.

/country: *value*

Specifies the language in which the print jobs are printed.

/customfile:" *text*"

Specifies the name of the file containing the custom language translation information. Use this option if you specify **/codepage:custom**.

/devicename:" *text*"

Specifies the name of the destination printer.

/duplex: *simplex* | *horizontal* | *vertical*

Specifies printing on two-sided paper.

/filterfile:" *text*"

The **/filterfile** is a defined programming API that allows you to pass the printer data stream to a third party or user supplied DLL. Enter text to signify where the printer data stream should go.

/font:" *text*"

Specifies the font to be used in the printer sessions. This can be any available font installed on the computer.

/formname: *string*

Specifies the name for the form.

/margin: *left, right, top, bottom*

Specifies the margins of the page in inches in the order of left, right, top, bottom. The default margins are 0".

/orientation: *portrait, landscape*

Specifies the page layout as portrait or landscape.

/papersize: *value*

Specifies the size of the paper for printing.

Valid Values and Descriptions for Papersize

Value	Constant Name (from wingdi.h)	Description

1	DMPAPER_LETTER	Letter, 8.5" x 11"
2	DMPAPER_LETTERSMALL	Letter Small, 8.5" x 11"
3	DMPAPER_TABLOID	Tabloid, 11" x 17"
4	DMPAPER_LEDGER	Ledger, 17" x 11"
5	DMPAPER_LEGAL	Legal, 8.5" x 14"
6	DMPAPER_STATEMENT	Statement, 5.5" x 8.5"
7	DMPAPER_EXECUTIVE	Executive, 7.25" x 10.5"
8	DMPAPER_A3	A3 Sheet, 297 x 420mm
9	DMPAPER_A4	A4 Sheet, 210 x 297mm
10	DMPAPER_A4SMALL	A4 Small Sheet, 210 x 297mm
11	DMPAPER_A5	A5 Sheet, 148 x 210mm
12	DMPAPER_B4	B4 Sheet, 250 x 354mm
13	DMPAPER_B5	B5 Sheet, 182 x 257mm
14	DMPAPER_FOLIO	Folio, 8.5" x 13"
15	DMPAPER_QUARTO	Quarto, 215 x 275mm
16	DMPAPER_10X14	10" x 14" Sheet
17	DMPAPER_11X17	11" x 17" Sheet
18	DMPAPER_NOTE	Note, 8.5" x 11"
19	DMPAPER_ENV_9	#9 Envelope, 3.875" x 8.875"
20	DMPAPER_ENV_10	#10 Envelope, 4.125" x 9.5"
21	DMPAPER_ENV_11	#11 Envelope, 4.5" x 10.375"
22	DMPAPER_ENV_12	#12 Envelope, 4.75" x 11"
23	DMPAPER_ENV_14	#14 Envelope, 5" x 11.5"
24	DMPAPER_CSHEET	C Sheet, 17" x 22"
25	DMPAPER_DSHEET	D Sheet, 22" x 34"
26	DMPAPER_ESHEET	E Sheet, 34" x 44"
27	DMPAPER_ENV_DL	DL Envelope, 110 x 220mm

28	DMPAPER_ENV_C5	C5 Envelope, 162 x 229mm
29	DMPAPER_ENV_C3	C3 Envelope, 324 x 458mm
30	DMPAPER_ENV_C4	C4 Envelope, 229 x 324mm
31	DMPAPER_ENV_C6	C6 Envelope, 114 x 162mm
32	DMPAPER_ENV_C65	C65 Envelope, 114 x 229mm
33	DMPAPER_ENV_B4	B4 Envelope, 250 x 353mm
34	DMPAPER_ENV_B5	B5 Envelope, 176 x 250mm
35	DMPAPER_ENV_B6	B6 Envelope, 176 x 125mm
36	DMPAPER_ENV_ITALY	Italy Envelope, 110 x 230mm
37	DMPAPER_ENV_MONARCH	Monarch Envelope, 3.875" x 7.5"
38	DMPAPER_ENV_PERSONAL	6.75 Envelope, 3.625" x 6.5"
39	DMPAPER_FANFOLD_US	US Standard Fanfold, 14.875" x 11"
40	DMPAPER_FANFOLD_STD_GERMAN	German Standard Fanfold, 8.5" x 12"
41	DMPAPER_FANFOLD_LGL_GERMAN	German Legal Fanfold, 8.5" x 13"

Additional values, supported by Windows

42	DMPAPER_ISO_B4	B4 (ISO) 250 x 353mm
43	DMPAPER_JAPANESE_POSTCARD	Japanese Postcard 100 x 148mm
44	DMPAPER_9X11	9" x 11"
45	DMPAPER_10X11	10" x 11"
46	DMPAPER_15X11	15" x 11"
47	DMPAPER_ENV_INVITE	Envelope Invite 220 x 220mm
48	DMPAPER_RESERVED_48	Reserved -- do not use
49	DMPAPER_RESERVED_49	Reserved -- do not use
50	DMPAPER_LETTER_EXTRA	Letter Extra, 9.275" x 12"
51	DMPAPER_LEGAL_EXTRA	Legal Extra, 9.275" x 15"
52	DMPAPER_TABLOID_EXTRA	Tabloid Extra, 11.69" x 18"
53	DMPAPER_A4_EXTRA	A4 Extra, 9.27" x 12.69"

54	DMPAPER_LETTER_TRANSVERSE	Letter Transverse, 8.275" x 11"
55	DMPAPER_A4_TRANSVERSE	A4 Transverse, 210 x 297mm
56	DMPAPER_LETTER_EXTRA_ TRANSVERSE	Letter Extra Transverse, 9.275" x 12"
57	DMPAPER_A_PLUS	SuperA / A4, 227 x 356mm
58	DMPAPER_B_PLUS	SuperB / A3, 305 x 487mm
59	DMPAPER_LETTER_PLUS	Letter Plus, 8.5" x 12.69"
60	DMPAPER_A4_PLUS	A4 Plus, 210 x 330mm
61	DMPAPER_A5_TRANSVERSE	A5 Transverse, 148 x 210mm
62	DMPAPER_B5_TRANSVERSE	B5 (JIS) Transverse, 182 x 257mm
63	DMPAPER_A3_EXTRA	A3 Extra, 322 x 445mm
64	DMPAPER_A5_EXTRA	A5 Extra, 174 x 235mm
65	DMPAPER_B5_EXTRA	A5 Extra, 174 x 235mm
66	DMPAPER_A2	A2, 420 x 594mm
67	DMPAPER_A3_TRANSVERSE	A3 Transverse, 297 x 420mm
68	DMPAPER_A3_EXTRA_TRANSVERSE	A3 Extra Transverse, 322 x 445mm

/paperlength: *value*

Specifies the length of the paper for printing. The values for paperlength and paperwidth are in tenths of a millimeter, and override the papersize setting.

/paperwidth:value

Specifies the width of the paper. The values for paperlength and paperwidth are in tenths of a millimeter, and override the papersize setting.

/pdtfile:" *text*"

Specifies the name of a printer definition file.

/provider:" *text*"

This parameter maps to one of the two print provider DLLs (PPD3270.DLL or PPD5250.DLL) and tells the Host Printer Service which DLL to load to do the printer communications (ppd3270.dll or ppd5250.dll). The text for **snacfg printsession3270** is PPD3270.

/printtofile:" *text*"

Specifies the name of a text file to which the output is to be sent. When information is entered into this parameter, the print job is saved into a file on the hard drive instead of being sent to the printer.

/quality: *high, medium, low, draft*

Specifies the quality of print.

/scalefactor: *value*

Specifies the amount by which the printed output is to be scaled from the physical page size, divided by 100. For example, a

scale factor of 50 would make the printed output half its normal size, 10 would be one tenth, etc.

/server:" *text*"

Specifies the name of the server to which the session should attach. This is usually the name of an Host Integration Server computer that is running the Host Print Service in the subdomain. When **/add** is used, this option is required.

/truetype: *bitmap, download, substitute*

Specifies TrueType fonts or substitutes.

/uniqueextension:{ yes | no }

When the **/printtofile** option is used, specifying YES enables the Host Print Service to generate a unique output file extension for each print job.

/overridefontsize:{ yes | no }

Specifies whether or not the default font size can be overridden.

/fontsize:*value*

Specifies the size of the font (in points) used for output.

/charset:*value*

Specifies the font face used for output.

/bypassgdi:{ yes | no }

This allows print jobs that are formatted with host software to bypass the Windows printing format system. All the data from the host is passed directly to the printer. The default is **no**.

/charsperline:*value*

Specifies the number of characters printed per line.

/feedperjob:{ yes | no }

If set to **yes**, it will issue a form feed between each print job. The default is **no**.

/feedignoreinitial:{ yes | no }

If set to **yes**, the beginning form feed in a print job will be ignored. The default is **no**.

/jobtermination:{ EndBracket | UnBind }

The print job will terminate when either a **When End Bracket Received** or a **When Unbind Received** command is received by the printer. The default is **EndBracket**.

/linespacing:*value*

Specifies the space (in points) between printed lines.

/linesperinch:{ 6 | 8 }

This specifies the number of lines per inch (either six or eight) in the print job. The default is **6**.

/linewrap:{ yes | no }

Specifies whether the text will automatically wrap at the end of a line.

/luname:" *text*"

The name of the configured 3270 printer LU. When **/add** is used, this option is required.

/monitorjob:{ yes | no }

If **yes** is specified, this feature sends a message to the host computer stating that the print job is completed. The default is **no**.

/pagewidth:{ 80 | 132 | 158 }

Specifies the default page width in number of characters per line. The values can be 80, 132, or 158 characters. The default is 80.

/skipblanklines:{ yes | no }

If there are blank lines in the print job, it will not print them if **yes** is specified. The default is **no**.

/timeout: *value*

Specifies the time limit, in seconds, for terminating print jobs.

/transparencyascii:{ yes | no }

This sets a flag that indicates that the transparent data from the host is in ASCII format and does not need translation from EBCDIC. The default is **no**.

/transparencycustom: *value*

Specifies the custom byte that sends the data stream in transparent mode. The IBM standard value is 0x35, but if the host print job uses another value, such as 0x36, then this should be specified.

See Also

Other Resources

Snacfg Reference

Snacfg PrintSessionAPPC

Purpose

Allows you to add, delete, modify, or view APPC print sessions defined in the Host Print Service.

Note
Configuration settings specified with snacfg printsessionAPPC correspond to local print server setting configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath]  [configpath] printsessionAPPCnameservernamelocalLUname
  [options]
   [configpath] printsessionAPPCname [options]
   [configpath] printsessionAPPCname
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of APPC print sessions.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a print session to the Host Print Service. To configure the print session, you must specify the server name and the configured APPC local LU alias name after the **/add**. The required options are **/server:***servername*and **/localLUalias:***localLUname*

/delete

Deletes the printer session. To delete the print session, you must specify the server name and the configured local LU alias after the **/delete**.

Options for APPC Print Sessions
/autoactivate:{ yes | no }

Specifies whether the printer session will automatically activate when Host Integration Server is started. The default is **yes**.

/bestfit:{ yes | no }

Specifies whether to scale the output to the paper size. The default is **yes**.

/codepage: {Country | Custom}

This defines the host code page language in which the print jobs are printed. The default is **Country** and the default language is **English (United States) [037]**. To change the default language, provide the number of the host code page of the country/region you want using the **/country** option.

If you want to use a custom file for the host code page, you must use **/customfile:***text,* where the text value is the name of the file containing the specifications.

/collate:{ yes | no }

Adds an option to collate pages sequentially.

/color:{ yes | no }

For color printers, printing will be gray scale if nothing is selected.

/comment:" *text*"

Adds an optional comment for the printer session. It can contain as many as 25 characters; enclose the comment in quotes.

/country: *value*

Specifies the language in which the print jobs are printed.

/customfile:" *text*"

Specifies the name of the file containing the custom language translation information. Use this option if you specify **/codepage:custom**.

/devicename:" *text*"

Specifies the name of the destination printer.

/duplex: *simplex* | *horizontal* | *vertical*

Specifies printing on two-sided paper.

/filterfile:" *text*"

This parameter is a defined programming API that allows you to pass the printer data stream to a third-party or user-supplied DLL. Enter the text of the file to which the printer data stream should go.

/font:" *text*"

Specifies the font to be used in the printer sessions. This can be any available font installed on the computer.

/formname: *string*

Specifies the name for the form.

/margin: *left, right, top, bottom*

Specifies the margins of the page, in inches, in the order of left, right, top, bottom. The default margins are 0".

/orientation: *portrait, landscape*

Specifies the page layout as portrait or landscape.

/pagewidth: *value (0-255)*

Specifies the width of page in characters.

/papersize: *value*

Specifies the size of the paper for printing.

/paperlength: *value*

Specifies the length of the paper for printing. The values for paperlength and paperwidth are in tenths of a millimeter, and override the papersize setting.

/paperwidth:v *alue*

Specifies the width of the paper. The values for paperlength and paperwidth are in tenths of a millimeter, and override the papersize setting.

/pdtfile:" *text*"

Specifies the name of a printer definition file.

/provider:" *text*"

This parameter maps to one of the two print provider DLLs (PPD3270.DLL or PPD5250.DLL) and tells Host Printer Service which DLL to load for the printer communications (ppd3270.dll or ppd5250.dll). The text for **snacfg printsessionappc** is PPD5250.

/printtofile:" *text*"

Specifies the name of a text file to which the output is to be sent. When information is entered into this parameter, the print

job is saved into a file on the hard drive instead of sending it to the printer.

/quality: *high, medium, low, draft*

Specifies the quality of print.

/scalefactor: *value*

specifies the amount by which the printed output is to be scaled from the physical page size, divided by 100. For example, a scale factor of 50 would make the printed output half its normal size, 10 would be one tenth, etc.

/server:" *text*"

Specifies the name of the server to which the session should attach. This is usually the name of an Host Integration Server computer that is running the Host Print Service in the subdomain. When **/add** is used, this option is required.

/systemtype: {AS400 | System36}

Specifies the type of system being used.

/truetype: *bitmap, download, substitute*

Specifies TrueType fonts or substitutes.

/uniqueextension:{ yes | no }

When the **/printtofile** option is used, the Host Print Service can generate a unique output file extension for each print job.

/overridefontsize:{ yes | no }

Specifies whether or not the default font size can be overridden.

/fontsize:*value*

Specifies the size of the font (in points) used for output.

/charset:*value*

Specifies the font face used for output.

/as400devicename:" *text*"

Specifies the name of the AS/400 computer that will be sending the print job.

/locallualias:" *text*"

Specifies the previously configured local LU alias. This is a required parameter when using **/add**.

/remotelualias:" *text*"

Specifies the default remote APPC LU. If an LU is specified here, do not enter text into **/remotefqname**.

/remotefqname:" *text*"

Specifies the fully qualified name or alias of a remote APPC LU. The fully qualified name is the network name and LU name. If text is entered here, do not enter text into **/remotelualias**.

/modename:" *text*"

The default mode name is QPCSUPP. The other mode names available are: #INTERSC, BLANK, and QSERVER.

/username:" *text*"

Specifies the authorized user.

/password:" *text*"

Specifies the user password.

/hostprinttransform:{ yes | no }

Enter *yes* to activate the Host Print Transform feature.

/msgqname:" *text*"

Specifies the qualified name of the message queue to which operational messages for this device are sent.

/msglibname:" *text*"

Specifies the library in which the message queue is located.

/hostprintertype:" *text*"

Specifies the type of host printer.

/copies:*value*

Specifies the number of copies to print.

/papersrc1:{ continuous | cut | autocut }

Specifies the type of paper used in the output device.

See Also

Other Resources

Snacfg Reference

Snacfg Server

Purpose

Allows you to view or change settings for servers. Can also be used to add a server, although the SNA Manager is the recommended interface for adding servers.

Note
Configuration settings specified with snacfg server correspond to server settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] servername [configpath] servername [options]
  [configpath] servername [options]
  [configpath] servername
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the servers on the local subdomain.

servername

Specifies the name of the server on which settings will be viewed or changed. The server name should be in the format machine_name or \\machine_name\snaservr (for specifying the primary node on the machine) and \\machine_name\snasrv02 (or snasrv03, snasrv04, etc.) for specifying the secondary nodes on the machine.

/add

Adds a Host Integration Server computer called *servername*. For adding a server, the recommended method is to use the SNA Manager, not **snacfg server**.

Before you can add connections to the server, link services must also be configured for the server. The recommended interface for configuring link services is the SNA Manager.

/delete

Deletes *servername*.

Options for Servers

/netname:" *text*"

Specifies the local network name, which identifies the SNA network of the local server. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. If the local network name is specified, the local control point name must also be specified (using the **/cpname** option).

/cpname:" *text*"

Specifies the local control point name, which identifies the local system to other control points (nodes) on the SNA network. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. If the local control point name is specified, the local network name must also be specified (using the **/netname** option).

When connecting to a host system and using a local control point name, the name should match the CPNAME = parameter in the host's PU definition.

/comment:" *text*"

Adds an optional comment for the specified server. The comment can contain as many as 25 characters; enclose the comments in quotes.

See Also

Other Resources

Snacfg Reference

Snacfg TN3Server

Purpose

Allows you to view, add, delete, or modify TN 3270 servers. This command can also be used to add a TN 3270 server, though the SNA Manager is the recommended interface for adding servers.

Note
Configuration settings specified with snacfg tn3server correspond to server settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] TN3Servername [configpath] TN3Servernname [configpath]
  TN3Servernname [options]
   [configpath] TN3Servername [options]
   [configpath] TN3Servername
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the TN 3270 servers on the local subdomain.

tn3servername

Specifies the computer name of the TN 3270 server on which settings will be viewed or changed.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a TN 3270 Server computer called *tn3servername*. For adding a server, the recommended method is to use the SNA Manager, not **snacfg tn3server**.

Before you can add connections to the server, link services must also be configured for the server. The recommended interface for configuring link services is the SNA Manager.

/delete

Deletes *tn3servername*.

Options for TN 3270 Servers

/logauditevents:{ yes | no }

Specifies whether the TN service uses the event log to log informational messages as well as error conditions. These are messages that log successful client connection and successful client termination.

/snaeventlog:{ yes | no }

Specifies whether the TN service uses the same event log as the SNA Manager. If this is set to *yes*, all TN3270 service event messages are written to the event log being used by the Host Integration Server system. If this is set to *no*, all TN3270 service event messages are written to the Windows Event Log on the local machine.

/nameresolution:{ yes | no }

Name resolution should only be selected if you are running a domain name resolver. A domain name resolver catalogs IP addresses and corresponding network names of connected computers. The domain name resolver allows you to enter the

name of a computer rather than the IP address when an IP address is required.

/closelistensocket:{ yes | no }

By default, the TN3270 Service always has a socket open for listening for incoming requests. If this option is turned on, the TN3270 Service stops listening on this socket once all of its defined LUs are in use. The purpose of this is to work with emulators that can try to connect to a number of computers running TN3270 Service and that connect to whichever computer accepts their connection attempt. In this case, it is useful if a computer with no LUs available is not listening.

/tn3270modeonly:{ yes | no }

The TN3270 Service now supports TN3270E, an enhancement to TN3270. When a client first connects to a computer running TN3270 Service, it negotiates which functions they both support. TN3270 emulators should be able to negotiate with TN3270 Service, if only to state that they do not support TN3270E. However, some TN3270 emulators are unable to negotiate properly with TN3270 Service, causing the negotiation to fail. For this reason the TN3270 Service has an option to default to TN3270 mode and not to use TN3270E features, so that these TN3270 negotiation problems do not occur.

/printerflowcontrol:{ yes | no }

If a TN3270 Server adheres strictly to the specification described in RFC 1647, there is no way of implementing flow control between a computer running TN3270 Service and a TN3270 client. In practice this causes no problems for display emulators, but it does cause a problem for printer emulators, which can be swamped with data and have no way of notifying the TN3270 Service that they cannot process any more messages. If this option is turned on, the TN3270 Service sends all messages to a TN3270 printer client as RESPONSE-REQUIRED, and does not send any messages until it has received a response for the previous message.

/idletimeout: *value 1-1440*

Specifies time limits in seconds. If the session is inactive for this length of time, then TN3270 Service disconnects the client.

/initstatusdelay: *value (0-86400)*

Specifies time limits. This is the delay between the time when TN3270 Service connects to a host session and the time the TN3270 Service starts updating the client screen. There are often a large number of startup messages when the TN3270 Service first connects to a host session, and this option gives the user the opportunity not to receive them all.

/msgclosedelay: *value (0-86400)*

Specifies time limits. When TN3270 Service forces a client to disconnect (for example, when the Host Integration Server session to the host has been lost), it sends the client an error message to be displayed on the screen. This value specifies the time between sending the message to the client and closing the socket with the client (which causes some clients to clear the screen, and so erase the message).

/refreshcycletime: *value (0-60)*

Specifies time limits. This is the delay between updates of the status on the display.

/inboundrusize: *value (256-32768)*

This controls the RU size (SNA message size) used by the TN3270 Service for logon messages to and from the host. The minimum value for inbound or outbound RU size is 256 bytes. If the host application sends large logon screens, these values should be increased.

/outboundrusize: *value (256-32768)*

This controls the RU size (SNA message size) used by the TN3270 Service for logon messages to and from the host. The minimum value for inbound or outbound RU size is 256 bytes. If the host application sends large logon screens, these values should be increased.

/portnumber: *value*

Specifies the port number associated with telnet. You can type another port number to override the default value for a given session. TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client to connect to multiple host computers.

/comment:" *text*"

Adds an optional comment for the specified server. The comment can contain as many as 25 characters; enclose the comments in quotes.

See Also

Other Resources

Snacfg Reference

Snacfg TN5Server

Purpose

Allows you to view, add, delete, or modify TN 5250 servers. the SNA Manager is the recommended interface for adding TN 5250 servers.

Note
Configuration settings specified with snacfg tn5server correspond to server settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] TN5Servername [configpath] TN5Servernname [configpath]
  TN5Servernname[options]
   [configpath] TN5Servername [options]
   [configpath] TN5Servername
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the TN 5250 servers on the local subdomain.

Tn5servername

Specifies the computer name of the TN 5250 server on which settings will be viewed or changed.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a Host Integration Server computer called *tn5servername*. For adding a TN 5250 server, the recommended method is to use the SNA Manager, not **snacfg tn5server**.

Before you can add connections to the server, link services must also be configured for the server. The recommended interface for configuring link services is the SNA Manager.

/delete

Deletes *tn5servername*.

Options for TN 5250 Servers
/portnumber: *value*

Specifies the port number associated with telnet (e.g., port number 23). You can type another port number to override the default value for a given session. TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client to connect to multiple host computers.

/comment:" *text*"

Adds an optional comment for the specified server. The comment can contain as many as 25 characters; enclose the comments in quotes.

See Also
Other Resources
Snacfg Reference

Snacfg TN3Session

Purpose

Allows you to view, add, delete, or modify TN 3270 sessions. the SNA Manager is the recommended interface for adding sessions.

Note
Configuration settings specified with snacfg tn3session correspond to session settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath]  luname [configpath] luname  [configpath] luname[option
  s]
   [configpath] luname [options]
   [configpath] luname
```

where

#configpath

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the sessions on the local subdomain.

luname

Specifies the LU name on which the session will be activated.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word snacfg, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a TN 3270 session called luname. For adding a session, the recommended method is to use the SNA Manager, not snacfg tn3session.

/delete

Deletes lu*name*.

Options for TN 3270 Sessions

/tn3server: *tn3servername*(required)

Specifies the computer name of the TN 3270 server on which settings will be viewed or changed. The *tn3servername* is required.

/assocname:" *text*"

Printer LUs can be marked as associated instead of generic or specific. To associate a printer LU with a terminal LU, type the Associated LU name. When this option is selected, the **terminal name** will default to IBM-3287-1. The quotes must be included.

/portnumber: *value*

Specifies the port number associated with telnet (e.g., port number 23). You can type another port number to override the default value for a given session. TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client to connect to multiple host computers.

/pooltype: *pooltype*

The values for pooltype are: GenericTerminal, SpecificTerminal, GenericPrinter, SpecificPrinter, AssocPrinter.

/maxsessions: *value*

Specifies the maximum number of LU-LU sessions that one independent LU can support. The limit set by maxsessions prevents an independent LU from using up too many unreserved session control blocks.

/modeltypes: *model*[,model,]

Values for TN3270 model types are:

Model2, Model3, Model4, Model5,

3275_2, 3276_2, 3277_2, 3278_2, 3278_2_E, 3279_2, 3279_2_E,

3276_3, 3278_3, 3278_3_E, 3279_3, 3279_3_E,

3276_4, 3278_4, 3278_4_E, 3279_4, 3279_4_E,

3278_5, 3278_5_E, 3279_5, 3279_5_E,

Dynamic, 3287_1

See Also

Other Resources

Snacfg Reference

Snacfg TN5Session

Purpose

Enables you to view, add, delete, or modify TN 5250 sessions. the SNA Manager is the recommended interface for adding sessions.

Note
Configuration settings specified with snacfg tn5session correspond to session settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] appcrlualias [configpath] appcrlualias [configpath] app
  crlualias[options]
   [configpath]  appcrlualias [options]
   [configpath] appcrlualias
```

where

#configpath

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the sessions on the local subdomain.

appcrlualias

Specifies the APPC remote LU alias name on which the session will be activated.

/print

Displays a list of the configuration settings of a print session. The displayed command does not contain the word snacfg, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a TN 5250 session called *appcrlualias*. For adding a session, the recommended method is to use the SNA Manager, not snacfg tn5session.

/delete

Deletes *appcrlualias*.

Options for TN 5250 Sessions
/tn5server: *tn5servername*(required)

Specifies the computer name of the TN 5250 server on which settings will be viewed or changed.

/locallu: *appclualias*(required)

Specifies the name of the APPC LU alias to view, add, modify, or delete. An APPC LU alias name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The APPC LU alias name cannot be the same as any other APPC LU alias name or pool name on the server.

/mode: *modename*

Specifies the name of the mode on which to carry out actions. A mode name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. The mode name cannot be the same as any other mode name in the subdomain of the server.

If no options are specified after modename, the configuration settings for the specified mode are displayed.

/user: *username*

Specifies the name of the user to view, change settings for, or delete. For adding TN 5250 users, the recommended method is to user the SNA Manager, not **snacfg tn5session**. If you are adding a user, *username* must match the name in the user's account on the Windows domain or on the local system.

/password: *password*

Specifies the current password for the user's account identified in *username*.

/modeltypes: *model*(,model,)

Specifies the model type. Values for TN5250 model types are:

5555_C01, 5555_B01, 3477_FC, 3477_FG

3180_2, 3179_2, 3196_A1

5292_2, 5291_1, 5251_11

/portnumber: *value*

Specifies the port number associated with telnet (e.g., port number 23). You can type another port number to override the default value for a given session. TN services listen on multiple ports simultaneously. You can set a default port number for the TN service (assign the port number to the server) and override this number on a per session basis (assign the port number to the LU session), allowing a single client to connect to multiple host computers.

/comment:" *text*"

Optionally, type a comment of up to 25 characters.

See Also
Other Resources
Snacfg Reference

Snacfg TNIPID

Purpose

Allows you to view, add, delete, or modify telnet IP (Internet Protocol) addresses. the SNA Manager is the recommended interface for adding TNIPIDs.

Note
Configuration settings specified with snacfg tnipid correspond to session settings configured with the SNA Manager.

Syntax

```
          [configpath]  [configpath] tnipid [configpath] tnipid  [configpath] tnipid [option
  s]
   [configpath] tnipid [options]
   [configpath] tnipid
```

where

#configpath

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the TNIPIDs on the local subdomain.

tnipid

Specifies the TN Internet Protocol (IP) address.

/print

Displays a list of the configuration settings of valid telnet IP addresses. The displayed command does not contain the word snacfg, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds a TNIPID called *tnipid*. For adding a *tnipid*, the recommended method is to use the SNA Manager, not snacfg tnipid.

/delete

Deletes *tnipid*.

Options for TNIPID

/session: *text*

Specifies the session name for the specified telnet IP address.

/subnet: *tnipaddress*

Specifies the TN Internet Protocol (IP) address. If a tnipid parses as "n.n.n.n," it will be treated as an IPADDRESS. Otherwise, it will be treated as an IPNAME.

See Also

Other Resources

Snacfg Reference

Snacfg User

Purpose

Enables you to view, modify settings for, or delete users recognized by Host Integration Server. Also lets you assign LUs or LU pools to 3270 users, or assign default LUs to 5250 users.

In addition, **snacfg user** enables you to add users to the list recognized by Host Integration Server. However, it is recommended that you use the SNA Manager, not **snacfg user**, when adding users.

Important
If you add user names by using the **/add** option with **snacfg user**, these names must match existing user account names in t he Windows domain, or a nonfunctioning configuration may result. To protect against errors with **snacfg user**, use the SNA Manager, instead.

Before assigning LUs or LU pools to users, you must first configure the LUs or LU pools.

Note
Configuration settings specified with snacfg user correspond to user or group settings configured with the SNA Manager.

Syntax

```
        [configpath]  [configpath] username [configpath] username [options]
   [configpath]  [configpath] username [options]
   [configpath] username
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of the 3270 users recognized by Host Integration Server.

domain/username

Specifies the domain and name of the user to add, view, change settings for, or delete.

For adding 3270 users, the recommended method is to use the SNA Manager, not **snacfg user**. If you are adding or removing a user, *username* must match the name in the user's account on the Windows domain specified by *domain*.

/add

Adds a user called *username* to the list of 3270 users recognized by Host Integration Server. The user must already have an account on the Windows domain or on the local system.

For adding 3270 users, the recommended method is to use the SNA Manager, not **snacfg user**.

/validate

Attempts to fill in any SIDs that are missing from user or group listings in the Host Integration Server configuration file. If the SIDs can be obtained from user accounts in the Windows domain for each user, the SIDs will be added to the listings in the Host Integration Server configuration.

Run **snacfg user /validate** after working offline and adding users to the configuration, so that the user listings in Host Integration Server will be functional.

/delete

Deletes *username*.

Options for Users of 3270 Emulators

/insert: *luname*[, *luname*,...]

Assigns the specified 3270 or LUA LUs or pools to the user. Separate multiple names with commas.

/remove: *luname*[, *luname*,...]

Deletes the assignment of 3270 or LUA LUs or pools to the user. Separate multiple names with commas.

/insertrlu: *luname*[, *luname*,...]

Assigns APPC Remote LUs to the user. Separate multiple names with commas. The name *rluname* can be either the RLU alias or its fully-qualified name of the form "*netname.luname*".

/removerlu: *luname*[, *luname*,...]

Deletes the assignment of APPC Remote LUs to the user. Separate multiple names with commas. The name *rluname* can be either the RLU alias or its fully-qualified name of the form "*netname.luname*".

/domain: *domainname*

Specifies the Windows domain in which the user account is found. For all actions involving the domain name associated with a user account, the recommended method is to use the SNA Manager, not **snacfg user**. If an incorrect domain name is specified with **snacfg user**, a nonfunctioning configuration may result.

/comment:" *text*"

Adds an optional comment for the specified user. The comment can contain as many as 25 characters. The quotes must be included.

/encrypt:{ yes | no }

Specify YES to enable data encryption between the server and client.

Options for Users of TPs or 5250 Emulators

/defloclu: *LUalias*

Specifies the default local APPC LU for this user or group. Both the user (or group) name and the LU alias must already be configured in Host Integration Server. If **/defloclu:** is typed without *LUalias*, the user or group record is cleared of any default local LU.

/defremlu: *LUalias*

Specifies the default remote APPC LU for this user or group. Both the user (or group) name and the LU alias must already be configured in Host Integration Server. If **/defremlu:** is typed without *LUalias*, the user or group record is cleared of any default remote LU.

See Also

Other Resources

Snacfg Reference

Snacfg Workstation

Purpose

The workstation assignment feature allows you to add, delete, modify, or view workstations. You can specify parameters such as IP address, workstation name, and define access parameters to the workstation.

Note
Configuration settings specified with snacfg workstation correspond to local workstation settings configured with the SNA M anager.

Syntax

```
          [configpath]  [configpath] workstationid [configpath] workstationid [configpath] w
  orkstationid [options]
   [configpath] workstationid [options]
   [configpath] workstationid
```

where

*configpath*

Specifies the path of the configuration file to view or change. If the configuration path is omitted, Host Integration Server will attempt to access the configuration file on the local system, using the path \Program Files\Host Integration Server\SYSTEM\CONFIG\COM.CFG.

/list

Generates a list of configured workstations.

/print

Displays a list of the configuration settings of a workstation. The displayed command does not contain the word **snacfg**, so that it can be redirected to a command file. Command files are discussed earlier in this section.

/add

Adds the workstation ID using a either a name or IP address to identify the workstation. The workstation ID is usually the workstation name.

If using an IP address to identify the workstation, the **/address** option is a required parameter.

To configure the workstation, either specify other options after **/add** or specify configuration options in additional **snacfg workstation** *workstationid* commands.

/delete

Deletes the workstation.

Options for Workstation
/insert: *luname*[,*luname,*]

Assigns the specified 3270 or LUA LUs or pools to the workstation. Separate multiple names with commas.

/remove: *luname*[,*luname,*]

Deletes the assignment of 3270 or LUA LUs or pools to the workstation. Separate multiple names with commas.

/insertrlu: *luname*[, *luname*,...]

Assigns APPC Remote LUs to the workstation. Separate multiple names with commas. The name *rluname* can be either the RLU alias or its fully-qualified name of the form "*netname.luname*".

/removerlu: *luname*[, *luname*,...]

Deletes the assignment of APPC Remote LUs to the workstation. Separate multiple names with commas. The name *rluname*

can be either the RLU alias or its fully-qualified name of the form "*netname.luname*".

/comment:" *comment*"

Sets the workstation's comment field. The quotes must be included.

/wkstaonly:{ yes | no }

If **yes**,it restricts resources available on the computer to the LUs assigned to the workstation. Users will not be able to access LUs assigned to them unless the LU is also assigned to the workstation. If **no**, it allows users access to the LUs assigned to the workstation and the LUs assigned to the user. The default is **no**.

/address:{ yes | no }

If **yes**, Host Integration Server will look for an IP address in place of an alphanumeric name for the workstation ID. This parameter must be set to **yes** if using an IP address. The default is **no**.

See Also

Other Resources

Snacfg Reference

Snacfg Error Messages

This section lists the error codes and their corresponding messages used by Snacfg commands.

Error Code	Message
3841	Insufficient privilege to run the SNA Manager program in this domain.
3842	The activation value is invalid.
3843	The link service is invalid.
3844	The allocation timeout is invalid.
3845	The audit level is invalid.
3846	The Control Point Name is invalid. The name can be from one through eight alphanumeric characters long.
3847	The Switched Connection Establishment Timeout is invalid. The range is from 10 through 500 seconds; the default is 300.
3848	The Contact Retry Limit is invalid. The range is from 1 through 20; the default is 10.
3849	The Contact Timeout is invalid. The range is from 5 (five-tenths of a second) through 300 (30 seconds). The default is 300 (30 seconds).

3850	The Partner Min Contention Winner Limit is invalid. The range is from 0 through the parallel session limit; the default is 0.
3851	The Minimum Contention Winner Limit is invalid. The range is from 0 through the parallel session limit; the default is 0.
3852	The default connection for the display verb is invalid.
3853	The DLC type is invalid.
3854	The domain name is invalid.
3855	The duplex value is invalid.
3856	Internal : Bad existing record.
3857	Facility Data is invalid. Facility Data can include as many as 126 hexadecimal digits.
3858	The configuration file is corrupt or does not have correct structure.
3859	The configuration file name is invalid. Verify that a primary or backup server is running in the domain you are trying to ad minister.
3860	Internal : Bad File Handle.

3861	The Idle Retry Limit is invalid. The range is from 1 through 255; the default is 10.
3862	The Idle Timeout is invalid. The range is from 1 (one-tenth of a second) through 300 (30 seconds). The default is 300 (30 seconds).
3863	The Automatic Activation Limit is invalid. The range is from 0 through the Minimum Contention Winner Limit.
3864	The Security Key Type is invalid.
3865	The Line Type is invalid.
3866	The LU Name is invalid. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. Names for APPC LUs must be different from one another, and names for non-APPC LUs must be different from one another.
3867	The LU Number is invalid. The range is from 1 through 255. The LU Number cannot be the same as that for any other LU using the connection, including local APPC LUs paired with remote APPC LUs that use the connection.
3869	The Mode Name is invalid. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase.
3870	The Display Model is invalid.
3871	The Name or Alias is invalid.
3872	The LU Name is invalid. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase. Names for APPC LUs must be different from one another.

3873	The NetView connection is invalid.
3874	The Network Name is invalid. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase.
3875	The Network Name is invalid. The name can be from one through eight characters long, and can contain alphanumeric characters and the special characters $, #, and @. Lowercase letters are converted to uppercase.
3876	The value for 'Model can be overridden' is invalid.
3877	The Packet Size is invalid. The possible values are 64, 128, 256, and 512. The default is 128.
3878	The LU Partner is invalid.
3879	The password is invalid. A password can contain from one through 10 characters.
3880	The Poll Address is invalid. The address is a two-digit hexadecimal number; the address cannot be the reserved values 00 or FF.
3881	The Poll Rate is invalid. The range is from 1 through 50 polls per second; the default is 5.
3882	The Poll Retry Limit is invalid. The range is from 1 through 255; the default is 10.
3883	The Poll Timeout is invalid. The range is from 1 (one-tenth of a second) through 300 (30 seconds). The default is 10 (1 second).

3884	The Port is invalid.
3885	Internal : Invalid record.
3886	The Remote X.25 Address is invalid. The address usually consists of 12 hexadecimal digits, but can contain up to 15 hexadecimal digits.
3887	The Role Type is invalid.
3888	The RTM Timers Run Until value is invalid.
3889	The first three digits of the Remote Node ID are invalid. These digits, also called the Block Number, can be any three hexadecimal digits except 000 or FFF, which are reserved.
3890	The last five digits of the Remote Node ID are invalid. These digits, also called the Node Number, can be any five hexadecimal digits.
3891	The Max Receive RU Size is invalid. The range is from 256 through 16384; the default is 1024.
3892	The Pacing Receive Count is invalid. The range is from 0 through 63; the default is 4.
3893	The Remote SAP Address is invalid. The address can be a 2-digit hexadecimal number that is a multiple of 04, between 04 and EC. The default is 04.
3894	The Security Key is invalid. For a Security Key in Hex, type a 16-digit hexadecimal number. For a Security Key in Characters, type an eight-character string consisting of alphanumeric characters and/or $, @, #, and the period (.).

3895	The server name is invalid.
3896	The Parallel Session Limit is invalid. Legal values are between 1 and 10000 inclusive.
3897	The Parallel Session Limit is invalid. Legal values are between 1 and 10000 inclusive.
3898	The Timeout for Starting TPs is invalid. The range is from 1 through 3600 seconds; the default is 60 seconds.
3899	The Timeout for Starting TPs is invalid. The range is from 1 through 3600 seconds; the default is 60 seconds.
3900	The Maximum BTU Length is invalid. The range is from 265 through 16393; the defaults are: Token Ring:\t1929Ethernet:\t1493Channel:\t4105SLDC:\t\t265X25:\t\t1033.
3901	The Remote Network Address is invalid. The address is a 12-digit hexadecimal value.
3902	The Retry Limit is invalid. The range is from 0 through 255; the default is 10. A value of zero means the system uses its internal default retry limit.
3903	The Receive ACK Threshold is invalid. The range is from 1 through 127; the default is 2.
3904	The Unacknowledged Send Limit is invalid. The range is from 1 through 127; the default is 8.
3905	The XID Retries value is invalid. The range is from 0 through 30; the default is 3.

3906	The first three digits of the Local Node ID are invalid. These digits, also called the Block Number, can be any three hexadecimal digits except 000 or FFF, which are reserved.
3907	The last five digits of the Local Node ID are invalid. These digits, also called the Node Number, can be any five hexadecimal digits except 00000, which is reserved.
3908	The Max Send RU Size is invalid. The range is from 256 through 16384; the default is 1024.
3909	Minimum Send RU is invalid.
3910	The Pacing Send Count is invalid. The range is from 0 through 63; the default is 4.
3911	User Data is invalid. User Data consists of an even number of hexadecimal charters, up to the maximum of 32 characters. The default is C3; this specifies the Qualified Logical Link Control (QLLC) protocol.
3912	The User ID is invalid. The User ID can contain from one through 10 characters.
3913	The PVC Alias is invalid. The range is from 1 through the number of configured PVC channels; the default is 1.
3914	The configuration file has been created by newer version of Host Integration Server. The Admin program only supports newer versions in read only mode.
3915	The Window Size is invalid.
3917	The connection cannot be modified when there are downstream LUs associated with it.

3918	There is already a 3270 LU with this name.
3919	There is already a 3270 Pool with this name.
3920	There is already a Link Service with this name.
3921	Multiple implicit partners or modes are not allowed for a local LU.
3922	There is already an LU with this LU Number.
3923	There is already an LUA LU with this name.
3924	There is already an LUA Pool with this name.
3925	The name or alias has already been used.
3926	There is already a Host Integration Server with this name.
3927	There is already a Downstream LU with this name.
3928	There is already an Downstream LU Pool with this name.

3929	The LU-LU pair already exists, or the adapter assignment already exists for the connection.
3930	There is already a user with this name.
3931	Unable to read the file.
3932	Unable to write the file.
3933	Field conflict. Unable to change.
3934	Writing to active file is not allowed.
3935	Configuration file was not found.
3936	Limit reached on objects associated with this item.
3937	Internal: Configuration is invalid.
3938	Session Limit conflicts with contention settings.
3939	Line Types do not match.

3940	Unable to write to active configuration file.
3941	Lost contact with remote domain.
3942	Unable to allocate memory.
3943	Internal: Memory corruption has occurred. Please reload configuration file.
3944	The number of LUs or connections for this Host Integration Server has been exceeded. Each Host Integration Server is capable of supporting 250 connections and 15000 dependent LUs.
3945	Internal: LUs are on different servers.
3946	Implicit partner or mode is not implicit.
3947	Internal: The item to be deleted is already absent from this grouping.
3948	Internal: Record has already been added to the configuration.
3949	A connection must have an associated link service.
3950	The Initially Active value must be zero for Local/Local Partners.

3951	The diagnostics record could not be found in the configuration file.
3952	Parallel Sessions are not supported on this LU.
3953	When a link service is to be used by multiple connections, and the connections accept incoming calls, the NRZ/NRZI settings for all the connections must match.
3954	The configuration file is an older config file.
3955	Internal: Target record already has an owner.
3956	The internal control record for the configuration file has exceeded its maximum size.
3957	During the conversion of an old Comm Server configuration file, COM.SEC was found to be corrupt, or of the wrong version type.
3958	Insufficient privilege or the configuration file is locked for read\\write access.
3959	Limit reached on number of objects in a configuration file.
3960	Limit reached on number of objects of this type.
3961	Internal: Invalid record type.

3962	Unable to change this record.
3963	The connection cannot be modified when there are LUs associated with it.
3964	The Session Limit is invalid.
3965	One of the DLC timeout values is invalid.
3966	This Return code is not used.
3967	The LU Alias has already been used. The LU Alias cannot be the same as any other APPC LU on its server.
3968	Unable to configure domain because the primary Host Integration Server for the domain is not active.
3969	The same Remote Network Address and SAP have already been specified for this link service in connection %s. To create more than one connection to the same remote machine, add one or more 802.2/DLC link services via Host Integration Server Setup over the same adapter with different local SAP addresses. This will automatically create additional connections in SNA Manager. Alternatively, if further SAPs are available at the remote machine, choose a different Remote SAP for this connection.
3970	No users or groups have been defined with Full Control Administration Access privileges. At least one user or group must be configured with permissions to alter the configuration.
3971	The Fully Qualified LU Name has already been used. The combination of the Net Name and LU Name cannot be the same as any other APPC LU on the server.
3972	Another Incoming Connection for this Link Service is already configured for XID 0. Multiple Incoming connections for one Link Service can only be configured if the connections use XID 3.

39733	Another incoming connection for this link service is configured for a different NRZ/NRZI setting. Multiple incoming connections for one link service must use the same NRZ/NRZI setting.
39734	Another Incoming Connection for this Link Service is configured for a different Poll Address. Multiple Incoming connections for one Link Service must use the same Poll Address.
39735	Another Primary Multidrop Connection for this Link Service is configured for this Poll Address. Primary Multidrop connections for the same Link Service must use different Poll Addresses.
39736	Another Primary Multidrop Connection for this Link Service is configured for a different NRZ/NRZI setting. Primary Multidrop connections for the same Link Service must use the same NRZ/NRZI setting.
39737	Another Primary Multidrop Connection for this Link Service is configured for a different Maximum BTU Length. Primary Multidrop connections for the same Link Service must use the same Maximum BTU Length.
39844	Another Primary Multidrop Connection for this Link Service is configured for a different XID Type. Primary Multidrop connections for the same Link Service must use the same XID Type.
39845	Another Primary Multidrop Connection for this Link Service is configured for a different Duplex setting. Primary Multidrop connections for the same Link Service must use the same Duplex setting.
39846	Another Primary Multidrop Connection for this Link Service is configured for a different Data Rate. Primary Multidrop connections for the same Link Service must use the same Data Rate.
39847	Another Primary Multidrop Connection for this Link Service is configured for a different Data Rate. Primary Multidrop connections for the same Link Service must use the same Data Rate.
39888	Another Primary Multidrop Connection for this Link Service is configured for a different Poll Timeout. Primary Multidrop connections for the same Link Service must use the same Poll Timeout.
39889	Another Primary Multidrop Connection for this Link Service is configured for a different Poll Retry Limit. Primary Multidrop connections for the same Link Service must use the same Poll Retry Limit.

3990	Another Primary Multidrop Connection for this Link Service is configured for a different Contact Timeout. Primary Multidrop connections for the same Link Service must use the same Contact Timeout.
3991	Another Primary Multidrop Connection for this Link Service is configured for a different Contact Retry Limit. Primary Multidrop connections for the same Link Service must use the same Contact Retry Limit.
3992	Leased SDLC connections cannot support Incoming Calls. Leased connections do not require either end to initiate the connection, therefore both ends treat the connection as Outgoing Calls only.
3993	The Partner LU specified for the CPI-C Symbolic Destination Name does not exist.
4000	The Mean time between broadcasts is invalid. The range is from 45 through 65535 seconds; the default is 60.
4001	No protocols have been defined to send Server Broadcasts. At least one protocol must be configured to send Server Broadcasts.
4002	The Channel Address is invalid.
4003	Could not perform the necessary privilege lookup to enable the required privileges in your process.
4004	Could not perform the necessary operations on the process token to enable the required privileges.
4005	You do not have the privileges required to set system ACLs on files.
4006	Unable to get or set security information on the config file.

4007	All connections on this SDLC Link Service must have their multidrop primary flag set the same way.
4008	Unable to obtain a write lock on the config file.
4009	The config file has become out of date and you must reload it.
4010	Bad parameter to function.
4011	The LU Alias has already been used. The LU alias cannot be the same as any other APPC LU on its server.
4012	You do not have the privileges necessary to lock the config file.
4013	Only a primary 3.0 or later computer can write a pre-3.0 configuration file.
4014	This configuration contains un-partnered dependent Local LU(s). These LU(s) have been discarded.
4015	The primary configuration file could not be located. The primary Host Integration Server may be unavailable. No configuration changes may be made at this time.
4016	This LU is already assigned to this user.
4017	There is already a dependent Local APPC LU with this LU number.

4018	There is already a 3270 LU with this LU number.
4019	Insufficient privilege to modify the config file.
4020	A 3270 LU name must be provided.
4021	The AS/400 device name must be provided.
4022	The Local LU Alias must be provide
4023	The Remote LU Alias or Fully Qualified Name must be provided.
4024	The mode must be provided.
4025	A Workstation with the same ID exists. Please use a different ID.
4026	The page width must be between 40 and 512.
4027	This LU is already assigned to this workstation.

See Also
Other Resources
Snacfg Reference

Linkcfg Reference

The following sections reference specific areas of Linkcfg.

This section contains:

- About Linkcfg
- Linkcfg Error Messages
- Demo SDLC Link Service
- Distributed Link Service
- DLC 802.2 Link Service

Linkcfg

Purpose

Allows you to install and delete link services from the command prompt. Linkcfg allows you to build script files to add link services. If you need to uninstall or reinstall Host Integration Server, you can reinstall link services by running the Linkcfg script files.

To view the list of link services you can install using Linkcfg, type the command linkcfg. The following appears:

```
LINKCFG [ LINKSVC /LSTYPE:{
"Andrew Twinax Link Service"|
"Atlantis/SAGEM SDLC Link Service"|
"Atlantis/SAGEM X.25 Link Service"|
"DCA ISCA SDLC Link Service"|
"DCA ISCA X.25 Link Service"|
"DEMO SDLC Link Service"
"Distributed Link Service"|
"DLC 802.2 Link Service"|
"IBM SDLC Link Service"|
"IBM Twinax Link Service"|
"IBM X.25 Link Service"|
"MicroGate SDLC Link Service"|
"MicroGate X.25 Link Service" } ]
 [ @commandfile ]
```

Note

If one of the link services was not copied during Setup, it will not appear on the list.

When setting the parameters for each of the link services, you must type the link service exactly as it appears above.

The configurable parameters for each link service are provided in the command-line help. To see the parameters for each, type the following:

LINKCFG LINKSVC /LSTYPE:"name of link service" where the name is one of those listed above. The available parameters will appear on the screen. The following sections list the available parameters for each link service.

See Also

Other Resources

Linkcfg Reference

Linkcfg Error Messages

The LinkCfg command utility returns the following codes and their corresponding messages. If you use the Windows Command Prompt, you only see the message. If you use Microsoft's System Management Server (SMS) to distribute link services across clients, you see the message and return code.

Error Code	Message
0	success.
160	command-line arguments are invalid.
1114	Unable to initialize Link Service DLL.
1157	Unable to locate DLL file.

In addition, LinkCfg will return error codes from GetLastError() calls for all file I/O operations.

See Also

Other Resources

Linkcfg Reference

Demo SDLC Link Service

Linkcfg options for the Demo SDLC link service. You can install the 3270 Continuous Demo, the 3270 File Transfer Demo, the 3270 Logon Demo, the AS400 Demo, and the LU1 and LU3 Printing Demos. Each of these demos is a script file that is run to demonstrate access to mainframe and AS/400 computers.

Note
Configuration settings specified with LINKCFG correspond to local link configuration settings in the SNA Manager.

Syntax

```
LINKCFG LINKSVC "title" (The quotes must be included.)
     /SERVER:servername
     /LSTYPE:"DEMO SDLC Link Service"
     /SCRIPTFILE: {"3270 Continuous Demo.dem"   |
                   "3270 File Transfer Demo.dem" |
                   "3270 Logon Demo.dem"         |
                   "AS400 Demo.dem"              |
                   "LU1 Printing Demo.dem"       |
                   "LU3 Printing Demo.dem"}
     /DISTRIBUTABLE:
```

/LINKSVC" *title*" (The quotes must be included.)

Enter the name of the service title. The quotes must be included. While any title name can be used, it is recommended that one of the following titles be used:

/LSTYPE:"Demo SDLC Link Service"

This parameter is used to list the available options for the Demo SDLC link service. You must type in the exact string as shown above for it to work correctly. The quotes must be included.

/SERVER: *servername*

This is the name of the computer on which the link service is to be installed.

/SCRIPTFILE:" *script file*"

This is the name of the demonstration script. Host Integration Server provides the following scripts:

- `"Continuous Demo.dem"` - this is the 3270 continuous demo script..
- `"File Transfer Demo.dem"` - this is the file transfer demo script.
- `"3270 Logon Demo.dem"` - this is the 3270 logon demo script.
- `"AS400 Demo.dem"` - this is the 5250 demo script.
- `"LU1 Printing Demo.dem` "- this is the demo script showing LU1 printing.
- `"LU3 Printing Demo.dem"` - this is the demo script showing LU3 printing. (The quotes must be included.)

See Also

Other Resources

Linkcfg Reference

Distributed Link Service

Linkcfg options for distributed link service.

Note
Configuration settings specified withlinkcfgcorrespond to local link configuration settings in the SNA Manager.

Syntax

```
LINKCFG LINKSVC "title"
     /SERVER:servername
     /LSTYPE:"Distributed Link Service"
     /REMOTELINKTYPE:{ Channel | Ethernet802.2 | LeasedSDLC |
                       SwitchedSDLC | TokenRing802.2 | Twinax | X.25QLLC }
     /REMOTELIST:\\server\service[;\\server\service;...]
     /ALTLIST:\\server\service[;\\server\service;...]
     /DOMAIN:domain
     /USERID:userid
     /PASSWORD:password
```

/LINKSVC" *title*"

Enter the name of the distributed link service. The quotes must be included. Any title name can be used.

/SERVER: *servername*

This is the name of the computer on which the link service is to be installed.

/LSTYPE:"Distributed Link Service"

This parameter is used to list the available options for the distributed link service. You must type in the exact string as shown above for it to work correctly. The quotes must be included.

/REMOTELINKTYPE:{ Channel | Ethernet 802.2 | LeasedSDLC | SwitchedSDLC | TokenRing802.2 | Twinax | X.25 }

This parameter defines the remote link type for the distributed link service.

/REMOTELIST:\\ *server*\ *service*[;\\ *server*\ *service*;]

This is the name of the remote link service, for example \\SNASERV1\SNADLC1;\\SNASERV2\SNADLC1.

/ALTLIST: \\server\service[;\\server\service;]

These are the names of alternate (backup) link services using the preceding format and example.

/DOMAIN: *domain*

Enter the name of the Host Integration Server subdomain. This is a required parameter if Host Integration Server is installed on a system account, not a local account.

/USERID: *userid*

Enter the name of the userid for the system account. This is a required parameter if Host Integration Server is installed on a system account, not a local account.

/PASSWORD: *password*

Enter the password of the system account. This is a required parameter if Host Integration Server is installed on a system account, not a local account.

See Also

Other Resources

Linkcfg Reference

DLC 802.2 Link Service

Linkcfg options for the DLC 802.2 link service.

Note
Configuration settings specified withlinkcfgcorrespond to local link configuration settings in the SNA Manager.

Recommended Syntax

```
LINKCFG LINKSVC "title" (The quotes must be included.)
     /SERVER:servername
     /LSTYPE:"DLC 802.2 Link Service"
     /ADAPTERTYPE:adaptertype
     /SAP:hexvalue
     /DISTRIBUTABLE:
```

/LINKSVC" *title*"

Enter the name of the service title. The quotes must be included. Any title name can be used.

/SERVER: *servername*

This is the name of the computer on which the link service is to be installed.

/LSTYPE:"DLC 802.2 Link Service"

This parameter is used to list the available options for the DLC 802.2 link service. You must type in the exact string as shown above for it to work correctly. The quotes must be included.

/ADAPTERTYPE: *adaptertype*

This parameter specifies the adapter type (using the adapter name) for the link service. Type the name of the adapter. This is an optional parameter and does not need to be defined if there is only one adapter installed. If there are multiple adapters in the computer and this parameter is not specified, it will use the first adapter it finds in the Registry, and only if the adapter has been bound with the DLC protocol.

The adapter name can be located in the registry using the following path:

```
HKEY_LOCAL_MACHINE
   SOFTWARE
      Microsoft
         Windows NT
            CurrentVersion
                NetworkCards
                   number
                      NetRules
```

The value is:

Bindform:REG_SZ: "*short name of adapter*" *yes yes container*

/SAP:hexvalue

Enter the SAP address using a hexadecimal value. For example, an acceptable value is 0x4, or 0xC.

/DISTRIBUTABLE:{ yes | no }

This parameter is used to specify whether the link service will be distributed. The Distributed Link Service feature provides a method for a Host Integration Server computer to connect with a host using a link service installed on a different Host Integration Server computer.

See Also

Other Resources

Linkcfg Reference

Sample Host Definitions

This section presents some sample host definitions used for establishing a Host Integration Server computer connection to IBM S/370 (host) systems. The samples show configuration values for Host Integration Server computer connections of the following types:

Connection type	Connection use	Host system
Token Ring	3270 emulation	Virtual Telecommunications Access Method (VTAM)
SDLC switched	3270 emulation	VTAM
SDLC leased	3270 emulation	VTAM
X.25	3270 emulation	VTAM
Any type	LU 6.2	Customer Information Control System (CICS)

Note

The VTAM listings in this topic omit the column 72-continuation character.

In This Section

Token Ring Definition

The following table shows a VTAM definition for a connection between a Host Integration Server computer and an IBM 9370 host computer across a Token Ring local area network (LAN).

VTAM Token Ring definitions

Name	Description	Parameters
LANODE	VBUILD	TYPE=LAN
R011100	PORT	CUADDR=100,MACADDR=400001701100, LANCON=(5,2),MAXDATA=2012, MAXSTN=32,SAPADDR=04
G011100	GROUP	DIAL=YES,ISTATUS=ACTIVE,LNCTL=SDLC
L011100A	LINE	CALL=INOUT
P011100A	PU	MAXLU=254
L011100B	LINE	CALL=INOUT
P011100B	PU	MAXLU=254
SWLAN	VBUILD	TYPE= SWNET,MAXGRP=5,MAXNO=12
SERVER01	PU	ADDR=C1,IDBLK=05D,IDNUM=00001, ANS=CONTINUE,MAXDATA=265, MAXOUT=7,MAXPATH=7,PACING=0, VPACING=0,SSCPFM=USSSCS, LANACK=(0,0),LANCON=(5,2), LANINACT=4.8,LANRESP=(5,2), LANSDWDW=(7,1),LANSW=YES, MACADDR=4000000000FD,PUTYPE=2, DISCNT=(NO),ISTATUS=ACTIVE, SAPADDR=04
E011100A	PATH	DIALNO=0004400001701100, GRPNM=G01100,GID=1,PID=1,USE=YES
T011100 00	LU	LOCADDR=002,DLOGMOD=D4C32782

T0 11 00 01	LU	LOCADDR=003,DLOGMOD=D4C32782
T0 11 00 02	LU	LOCADDR=004,DLOGMOD=D4C32782
T0 11 00 03	LU	LOCADDR=005,DLOGMOD=D4C32782
T0 11 00 04	LU	LOCADDR=006,DLOGMOD=LU33286S

The following table defines key parameters for Token Ring.

VTAM parameter descriptions for Token Ring

VTAM parameter	Corresponding Host Integration Server parameter	Description
IDBLK=	First three digits of local node ID (basic connection setting)	A three-digit hexadecimal value that, together with IDNUM, identifies the Host Integration Server computer. The values 000 and FFF cannot be used; these values are reserved.
IDNUM=	Last five digits of local node ID (basic connection setting)	A five-digit hexadecimal value that, together with IDBLK, identifies the Host Integration Server computer.
MAXDATA= (in the PU definition for SERVER01)	Max basic transmission unit (BTU) Length (advanced 802.2 setting)	The maximum length of a BTU sent or received; BTUs are also known as I-frames. Max BTU Length should be less than or equal to MAXDATA.
MACADDR= (in the PORT definition)	Remote Network Address (basic 802.2 setting)	The media access control address, a 12-digit hexadecimal address.
SAPADDR= (in the PU definition)	Remote service access point (SAP) Address (advanced 802.2 setting)	The system access point address, a two-digit hexadecimal number, a multiple of 4, between 04 and EC.
LOCADDR= (in the LU definition)	Local unit (LU) Number	The LU local address at the physical unit (PU). For independent LUs that communicate with a host, the LOCADDR parameter must be set to 0. For dependent LUs with VTAM, the LOCADDR must be at least 2. Also, for dependent LUs, the LU Numbers set in the Host Integration Server computer must match the LOCADDR values on the host.

The GRPNM value shown in the sample above is a group name; group names must match the label of a VTAM group definition. The DLOGMOD values shown in the sample are the logon mode table entries from the mode table. For descriptions of the group macros used in these samples and a complete listing of default mode table entries, see the documentation for your IBM product.

See Also

Other Resources

Sample Host Definitions

SDLC Definition (for Switched and Leased Lines)

The following table shows a sample IBM 9370 VTAM definition for four Synchronous Data Link Control (SDLC) lines. The LLGROUP label shows the VTAM definitions for lines B80 and BB0 of a leased group. The SNAGROUP label shows the VTAM definitions for lines B90 and BA0 of a switched group.

VTAM definition for four SDLC lines

Label	Operation	Operands
IBMLINE	VBUILD	TYPE=CA
LLGROUP	GROUP	LNCTL=SDLC,DIAL=NO,ACTIVTO=10
L01B80	LINE	ADDRESS=B80,PAUSE=0.4,REPLYTO=2.0, RETRIES=7
P01B80A	PU	ADDR=C1,DISCNT=NO,PASSLIM=7, PUTYPE=2,MAXOUT=7,USSTAB=AUSSTAB
T01B8002	LU	LOCADDR=002,DLOGMOD=D4C32792
T01B8003	LU	LOCADDR=003,DLOGMOD=D4C32792
T01B8004	LU	LOCADDR=004,DLOGMOD=D4C32792
L01BB0	LINE	ADDRESS=BB0,PAUSE=0.4,REPLYTO=2.0, RETRIES=7
P01BB0A	PU	ADDR=C1,DISCNT=NO,PASSLIM=7, PUTYPE=2,MAXOUT=7,USSTAB=AUSSTAB
T01BB000	LU	LOCADDR=002,DLOGMOD=D4C32792
T01BB001	LU	LOCADDR=003,DLOGMOD=D4C32792
T01BB002	LU	LOCADDR=004,DLOGMOD=D4C32792
T01BB003	LU	LOCADDR=005,DLOGMOD=D3C32792
T01BB004	LU	LOCADDR=006,DLOGMOD=LU33286S
SNAGROUP	GROUP	LNCTL=SDLC,DIAL=YES,MAXLU=20
L01B90	LINE	ADDRESS=B90,PAUSE=0.4,REPLYTO=2.0, RETRIES=7
P01B90A	PU	
L01BA0	LINE	ADDRESS=BA0,PAUSE=0.4,REPLYTO=2.0, RETRIES=7
P01BA0A	PU	

The following table shows the sample IBM 9370 VTAM switched definition for the SNAGROUP label.

VTAM switched definition for SNAGROUP

Label	Operation	Operands
SWNODE	VBUILD	TYPE=SWNET,MAXGRP=0,MAXNO=0

SWPU1	PU	ADDR=C1,IDBLK=017,IDNUM=DC006, DISCNT=NO,PASSLIM=2,PUTYPE=2, MAXDATA=265,MAXOUT=2, USSTAB=MSUSSTAB
T01BA000	LU	LOCADDR=2,DLOGMOD=D4C32782
T01BA001	LU	LOCADDR=3,DLOGMOD=D4C32782
T01BA002	LU	LOCADDR=4,DLOGMOD=D4C32782
T01BA003	LU	LOCADDR=5,DLOGMOD=D4C32782
T01BA004	LU	LOCADDR=6,DLOGMOD=LU33286S

The following table describes key parameters for SDLC.

VTAM Parameter Descriptions for SDLC

VTAM parameter	Corresponding Host Integration Server parameter	Description
ADDR=	Poll Address (advanced SDLC setting)	A two-digit hexadecimal value that identifies the control unit that the host uses to poll.
IDBLK=	First three digits of Local Node ID (basic connection setting)	A three-digit hexadecimal value that, together with IDNUM, identifies the Host Integration Server computer. The values 000 and FFF cannot be used; these values are reserved.
IDNUM=	Last five digits of Local Node ID (basic connection setting)	A five-digit hexadecimal value that, together with IDBLK, identifies the Host Integration Server computer.
MAXDATA= (in the PU definition for SWPU1)	Max basic transmission unit (BTU) Length (advanced SDLC setting)	The maximum length of a BTU sent or received; BTUs are also known as I-frames.
LOCADDR=	LU Number	The LU local address at the PU. For independent LUs that communicate with a host, the LOCADDR parameter must be set to zero. For dependent LUs with VTAM, the LOCADDR must be at least 2. Also, for dependent LUs, the LU Numbers set in the Host Integration Server computer must match the LOCADDR values on the host.

The DLOGMOD values shown in the example are the logon mode table entries from the mode table. For a complete listing of default mode table entries, see the documentation for your IBM product.

See Also

Other Resources

Sample Host Definitions

X.25 Definition

The following table shows a sample IBM 9370 VTAM definition for X.25 circuits. The X25GROUP label shows the VTAM definitions for switched virtual circuits 001 through 006.

VTAM definition for X.25 connection

Label	Operation	Operands
X25NODE	VBUILD	TYPE=PACKET
PORTA00	PORT	CUADDR=(A00,A08),NETTYPE=1, CHARGACC=YES,CHARGE=NO, VCALLS=(,,001,006,,),DIALNO=31370023061 MAXOUT=7,NETLEVEL=80,PMOD=8, PLENGTH=256,PWINDOW=3,REPLYT0=3, RETRIES=7
VCPA00	VCPARMS	LC=(1,6),PWINDOW=3,PLENGTH=256
X25GROUP	GROUP	DIAL=YES,LNCTL=SDLC,CALL=INOUT, ISTATUS=ACTIVE
L01A00A	LINE	ADDRESS=001,AUTO=001, ISTATUS=ACTIVE
P01AOOA	PU	MAXLU=32
L01A00B	LINE	ADDRESS=002,ISTATUS=INACTIVE, AUTO=002
P01AOOB	PU	MAXLU=32
L01A00C	LINE	ADDRESS=003,ISTATUS=INACTIVE, AUTO=003
P01AOOC	PU	MAXLU=32
L01A00D	LINE	ADDRESS=004,ISTATUS=INACTIVE, AUTO=004
P01AOOD	PU	MAXLU=32
L01A00E	LINE	ADDRESS=005,ISTATUS=INACTIVE, AUTO=005
P01AOOE	PU	MAXLU=32
L01A00F	LINE	ADDRESS=006,ISTATUS=INACTIVE, AUTO=006
P01AOOF	PU	MAXLU=32

The following table shows the sample IBM 9370 VTAM switched definition for the X25GROUP label.

VTAM switched definition for X25GROUP

Label	Operation	Operands
X25	VBUILD	TYPE=SWNET,MAXGRP=5,MAXNO=12
USER01	PU	ADDR=C1,IDBLK=05D,IDNUM=00025, ANS=CONTINUE,MAXDATA=265, MAXOUT=7,MAXPATH=0,PACING=0, VPACING=0,SSCPFM=USSSCS, IRETRY=YES,USSTAB=MSUSSTAB, PUTYPE=2,DISCNT=YES,ISTATUS=ACTIVE
T01A0002	LU	LOCADDR=002,DLOGMOD=D4C32792
T01A0003	LU	LOCADDR=003,DLOGMOD=D4C32792
T01A0004	LU	LOCADDR=004,DLOGMOD=D4C32792
T01A0005	LU	LOCADDR=005,DLOGMOD=D4C32792
T01A0006	LU	LOCADDR=006,DLOGMOD=LU33286S

The following table defines key parameters for X.25.

VTAM Parameter Descriptions for X.25

VTAM parameter	Corresponding Host Integration Server parameter	Description
IDBLK=	First three digits of Local Node ID (basic connection setting)	A three-digit hexadecimal value that, together with IDNUM, identifies the Host Integration Server computer. The values 000 and FFF cannot be used; these values are reserved.
IDNUM=	Last five digits of Local Node ID (basic connection setting)	A five-digit hexadecimal value that, together with IDBLK, identifies the Host Integration Server computer.
DIALNO= (in the PORT definition)	Remote X.25 Address	An address that identifies an X.25 system. The address consists of from 1 through 15 hexadecimal digits. If a 15-digit address is used, the final 3 digits are used for routing between stations with the same 12-digit address.

MAXDATA= (in the PU definition for USER01)	Max BTU Length (advanced X.25 setting)	The maximum length of a BTU sent or received.
LOCADDR=	LU Number	The LU local address at the PU. For independent LUs that communicate with a host, the LOCADDR parameter must be set to 0. For dependent LUs with VTAM, the LOCADDR must be at least 2. Also, for dependent LUs, the LU numbers set in the Host Integration Server computer must match the LOCADDR values on the host. In Table A.4, the LU numbers range from 002 to 006.

See Also

Other Resources

Sample Host Definitions

CICS and VTAM Sample Definitions for LU 6.2

This topic presents sample definitions for CICS (version 1.6) and VTAM (version 3.2) for Host Integration Server computer connections to an IBM host system for LU 6.2 operation.

To configure a CICS host system for LU 6.2 operation with a Host Integration Server system, you configure the following information at the host:

- The remote systems with which CICS is able to communicate.
- The name used by VTAM to communicate with each remote system.
- The transaction programs (TPs) available on the CICS system.
- The programming language (PL/I, COBOL, or Assembler) in which each program is written.

This information is configured in CICS and VTAM tables. This topic provides sample definitions that show how to configure CICS for LU 6.2 using these CICS and VTAM tables.

In This Section

CICS Tables

CICS uses the following tables to define LU 6.2 information:

- Terminal Control Table (TCT)—defines the remote systems to CICS.
- Program Control Table (PCT)—defines the local TPs to CICS.
- Program Processing Table (PPT)—defines the load module characteristics to CICS.

The following sections describe these tables, and include sample definitions to illustrate how they are used.

Note
These samples assume there are four TPs residing on CICS written in PL/I with the following names: TRAN0, TRAN1, TRAN2, and TRAN3.

This section contains:

Terminal Control Table

The sample Terminal Control Table (TCT) defines the remote systems to CICS. Each remote system is specified using a DFHTCT definition, which includes both the name by which CICS knows the system (the SYSIDNT field), and the name by which VTAM knows the system (the NETNAME field).

Sample CICS terminal control table entries

Label	Definition	Operands
TP11	DFHTCT	TYPE=SYSTEM ACCMETH=VTAM, FEATURE=SINGLE,TRMTYPE=LUTYPE62, NETNAME=TPLU6207,SYSIDNT=DC11, MODENAM=LU62,CONNECT=AUTO, BUFFER=256,RUSIZE=256, TRMSTAT=(TRANSCEIVE)
TP12	DFHTCT	TYPE=SYSTEM ACCMETH=VTAM, FEATURE=SINGLE,TRMTYPE=LUTYPE62, NETNAME=TPLU6208,SYSIDNT=DC12, MODENAM=LU62,CONNECT=AUTO, BUFFER=256,RUSIZE=256, TRMSTAT=(TRANSCEIVE)
TP13	DFHTCT	TYPE=SYSTEM ACCMETH=VTAM, FEATURE=SINGLE,TRMTYPE=LUTYPE62, NETNAME=TPLU6209,SYSIDNT=DC13, MODENAM=LU62,CONNECT=AUTO, BUFFER=256,RUSIZE=256, TRMSTAT=(TRANSCEIVE)
TP14	DFHTCT	TYPE=SYSTEM ACCMETH=VTAM, FEATURE=SINGLE,TRMTYPE=LUTYPE62, NETNAME=TPLU620A,SYSIDNT=DC14, MODENAM=LU62,CONNECT=AUTO, BUFFER=256,RUSIZE=256, TRMSTAT=(TRANSCEIVE)

See Also
Other Resources
CICS Tables

Program Control Table

The Program Control Table (PCT) defines the local transaction programs (TPs) to CICS. Each TP is specified using a DFHPCT definition that includes the name by which the TP is invoked (the TRANSID field) and the corresponding load module name from the CICS library of TPs.

Sample CICS program control table entries

TP Label	Definition	Operands
TRAN0	DFHPCT	TYPE=ENTRY PROGRAM=TRAN0000, TWASIZE=1000,SPURGE=NO,TPURGE=NO, RSL=00,RSLC=NO,MODENAM=NORMAL, TRANSID=TRAN0
TRAN1	DFHPCT	TYPE=ENTRY PROGRAM=TRAN0001, TWASIZE=1000,SPURGE=NO,TPURGE=NO, RSL=00,RSLC=NO,MODENAM=NORMAL, TRANSID=TRAN1
TRAN2	DFHPCT	TYPE=ENTRY PROGRAM=TRAN0002, TWASIZE=1000,SPURGE=NO,TPURGE=NO, RSL=00,RSLC=NO,MODENAM=NORMAL, TRANSID=TRAN2
TRAN3	DFHPCT	TYPE=ENTRY PROGRAM=TRAN0003, TWASIZE=1000,SPURGE=NO,TPURGE=NO, RSL=00,RSLC=NO,MODENAM=NORMAL, TRANSID=TRAN3

* ****** * PROFILE TO BE USED IN LU6.2 ALLOCATE * ****** *

TP Label	Definition	Operands
	DFHPCT	TYPE=PROFILE, PROFILE=DFHCICSA, INBFMH=ALL,MODENAME=LU62

See Also

Other Resources

CICS Tables

Program Processing Table

The Program Processing Table (PPT) defines the load module characteristics to CICS.

Each load module is defined using the DFHPPT definition, which includes the name of the load module (the PROGRAM field), and the language in which the module has been written. Examples are PL/I, Assembler, or COBOL (the PGMLANG field).

Sample CICS program processing table entries

Definition	Program name	Language
DFHPPT TYPE=ENTRY	PROGRAM=TRAN0000	PGMLANG=PL/I
DFHPPT TYPE=ENTRY	PROGRAM=TRAN0001	PGMLANG=PL/I
DFHPPT TYPE=ENTRY	PROGRAM=TRAN0002	PGMLANG=PL/I
DFHPPT TYPE=ENTRY	PROGRAM=TRAN0003	PGMLANG=PL/I

See Also

Other Resources

CICS Tables

VTAM Definitions

Each remote system must be defined to VTAM in its table of Network Addressable Units, (NAUs), using logical unit (LU) definitions.

If a LOGAPPL parameter is specified on the LU definition naming the CICS system, CICS will attempt to establish a session to that LU whenever it becomes active. This is necessary if TPs residing in CICS do not use the **ALLOCATE** verb with RETURN_CONTROL=WHEN_SESSION_ALLOCATED to establish sessions. There must also be a VTAM mode table definition present for LU 6.2, although CICS does not use the values specified here.

Sample VTAM network control program (NCP) definition for LU 6.2

Label	Operation	Operands
TPPU62	PU	ADDR=C1,IBLK=03E,IDNUM=01A1A, IRETRY=YES,DISCNT=NO,ISTATUS=ACTIVE, MAXDATA=265,SSCPFM=USSSCS, MODETAB=MRNDS,USSTAB=USSNDN1, MAXOUT=7,PASSLIM=7,PUTYPE=2
TPLU6207	LU	LOCADDR=7,DLOGMOD=LU62, USSTAB=USSNOMSG
TPLU6208	LU	LOCADDR=8,DLOGMOD=LU62, USSTAB=USSNOMSG
TPLU6209	LU	LOCADDR=9,DLOGMOD=LU62, USSTAB=USSNOMSG
TPLU620A	LU	LOCADDR=A,DLOGMOD=LU62, USSTAB=USSNOMSG

The following table shows NCP mode table entries for LU 6.2.

Sample NCP mode table entries for LU 6.2

Label	Operation	Operands
	MODETAB	
LU62	MODEENT	LOGMODE=NORMAL
	. . .	
	MODEEND	

Note

It is recommended that the mode used by LU 6.2 be listed first, as shown by the LU62 example in the preceding table.

See Also

Concepts

TP Coding Requirements

TP Coding Requirements

To work with CICS, TPs must be coded to specify a SYSID parameter on the **ALLOCATE** request that matches the defined SYSIDNT parameter configured in the DFHTCT tables. This request must be in the following format:

```
EXEC CICS ALLOCATE SYSID('DC11')
```

The SYSID field must match the SYSIDNT field specified for a CICS program by the DFHTCT definition in the TCT (Terminal Control Table). The TCT then uses the NETNAME parameter to refer to a specific LU in the VTAM table of the NAUs (Network Addressable Units). This parameter is used by the TP to communicate with the remote system.

See Also

Reference

VTAM Definitions

Sense Codes

This section lists the sense codes used by Host Integration Server.

Host Integration Server sense codes are sent on either a negative response to an outbound data stream or an LUSTAT that notifies the host of a change in the secondary logical unit (LU) state. The sense codes generated by a 3270 emulator program are shown in the following tables.

Negative response sense codes

Value	Meaning
0802	Intervention required (LU 1/LU 3 printer soft error).
0814	Bracket bid reject RTR forthcoming.
081B	Receiver in transmit mode.
081C	Request not executable (hard error).
0821	Invalid session parameters (negotiable BIND request).
0829	Change direction required.
082B	Presentation space integrity lost.
082D	LU busy (usually local copy print in progress).
082E	Intervention required (local copy print soft error).
082F	Request not executable (local copy print hard error).
0843	WCC print command bit not sent (RQD or RQE, CD, EB).
0863	Referenced character set does not exist.
0871	Read partition state error (SLU in retry state).
1003	Function not supported.
1005	Parameter modifying a control function is invalid.
1007	Category not supported (SSCP data for host printer).

LUSTAT sense codes

Value	Meaning
0001 B000	Host initialized local copy soft error recovery.
081C 0000	Soft error changed to hard error (LU 1/LU 3).
081C B000	Soft error changed to hard error (local copy).
082B 0000	LU session changed from SSCP to PLU (SYSREQ).

See Also

Other Resources

Administrator's Reference

Character Tables

This section provides tables that show the ASCII, ANSI, and IBM extended character sets.

ASCII Character Set

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`
^A	1	01	☺	SOH	33	21	!	65	41	A	97	61	a
^B	2	02	☻	STX	34	22	"	66	42	B	98	62	b
^C	3	03	♥	ETX	35	23	#	67	43	C	99	63	c
^D	4	04	♦	EOT	36	24	$	68	44	D	100	64	d
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e
^F	6	06	♠	ACK	38	26	&	70	46	F	102	66	f
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g
^H	8	08	◘	BS	40	28	(	72	48	H	104	68	h
^I	9	09	○	HT	41	29	)	73	49	I	105	69	i
^J	10	0A	◙	LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B	♂	VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D	♪	CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E	♫	SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F	☼	SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10	►	DLE	48	30	0	80	50	P	112	70	p
^Q	17	11	◄	DC1	49	31	1	81	51	Q	113	71	q
^R	18	12	↕	DC2	50	32	2	82	52	R	114	72	r
^S	19	13	‼	DC3	51	33	3	83	53	S	115	73	s
^T	20	14	¶	DC4	52	34	4	84	54	T	116	74	t
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u
^V	22	16	▬	SYN	54	36	6	86	56	V	118	76	v
^W	23	17	↨	ETB	55	37	7	87	57	W	119	77	w
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A	→	SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B	←	ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C	∟	FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D	↔	GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂†

† ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL+BKSP key.

ASCII Character Set (continued)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

ANSI Character Set

0	■	32		64	@	96	`	128	■	160		192	À	224	à
1	■	33	!	65	A	97	a	129	■	161	¡	193	Á	225	á
2	■	34	"	66	B	98	b	130	‚	162	¢	194	Â	226	â
3	■	35	#	67	C	99	c	131	ƒ	163	£	195	Ã	227	ã
4	■	36	$	68	D	100	d	132	„	164	¤	196	Ä	228	ä
5	■	37	%	69	E	101	e	133	…	165	¥	197	Å	229	å
6	■	38	&	70	F	102	f	134	†	166	¦	198	Æ	230	æ
7	■	39	'	71	G	103	g	135	‡	167	§	199	Ç	231	ç
8	■	40	(	72	H	104	h	136	ˆ	168	¨	200	È	232	è
9	■	41	)	73	I	105	i	137	‰	169	©	201	É	233	é
10	■	42	*	74	J	106	j	138	Š	170	ª	202	Ê	234	ê
11	■	43	+	75	K	107	k	139	‹	171	«	203	Ë	235	ë
12	■	44	,	76	L	108	l	140	Œ	172	¬	204	Ì	236	ì
13	■	45	-	77	M	109	m	141	■	173	-	205	Í	237	í
14	■	46	.	78	N	110	n	142	■	174	®	206	Î	238	î
15	■	47	/	79	O	111	o	143	■	175	¯	207	Ï	239	ï
16	■	48	0	80	P	112	p	144	■	176	°	208	Ð	240	ð
17	■	49	1	81	Q	113	q	145	‘	177	±	209	Ñ	241	ñ
18	■	50	2	82	R	114	r	146	’	178	²	210	Ò	242	ò
19	■	51	3	83	S	115	s	147	“	179	³	211	Ó	243	ó
20	■	52	4	84	T	116	t	148	”	180	´	212	Ô	244	ô
21	■	53	5	85	U	117	u	149	•	181	µ	213	Õ	245	õ
22	■	54	6	86	V	118	v	150	–	182	¶	214	Ö	246	ö
23	■	55	7	87	W	119	w	151	—	183	·	215	×	247	÷
24	■	56	8	88	X	120	x	152	˜	184	¸	216	Ø	248	ø
25	■	57	9	89	Y	121	y	153	™	185	¹	217	Ù	249	ù
26	■	58	:	90	Z	122	z	154	š	186	º	218	Ú	250	ú
27	■	59	;	91	[	123	{	155	›	187	»	219	Û	251	û
28	■	60	<	92	\	124	|	156	œ	188	¼	220	Ü	252	ü
29	■	61	=	93	]	125	}	157	■	189	½	221	Ý	253	ý
30	■	62	>	94	^	126	~	158	■	190	¾	222	Þ	254	þ
31	■	63	?	95	_	127	■	159	Ÿ	191	¿	223	ß	255	ÿ

■ Indicates that this character isn't supported by Windows.

ᵀᵀ Indicates that this character is available only in TrueType fonts.

IBM Extended Character Set

0		32		64	@	96	`	128	Ç	160	á	192	└	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
9	○	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	♪	45	_	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	☼	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	►	48	0	80	P	112	p	144	É	176	░	208	ð	240	-
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	=
19	‼	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	▬	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	ø	187	╗	219	▉	251	¹
28	∟	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	↔	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

Code Page 850 - multilingual (Latin)

See Also

Reference

Host Print Service Character Translation Table Format

Host Print Service Character Translation Table Format

The character translation table that can be used by the custom code page option of Host Print service is a 512-byte file, split into two 256-byte regions. Bytes 0255 are the mapping bytes for data from the host; bytes 256511 map data to the host.

For example, the following illustrates standard translation tables for the 037 EBCDIC code page:

```
   | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
---------------------------------------------------
00| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
10| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
30| 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
40| 20 a0 e2 e4 e0 e1 e3 e5 e7 f1 a2 2e 3c 28 2b 7c
50| 26 e9 ea eb e8 ed ee ef ec df 21 24 2a 29 3b ac
60| 2d 2f c2 c4 c0 c1 c3 c5 c7 d1 a6 2c 25 5f 3e 3f
70| f8 c9 ca cb c8 cd ce cf cc 60 3a 23 40 27 3d 22
80| d8 61 62 63 64 65 66 67 68 69 ab bb f0 fd de b1
90| b0 6a 6b 6c 6d 6e 6f 70 71 72 aa ba e6 b8 c6 a4
a0| b5 7e 73 74 75 76 77 78 79 7a a1 bf d0 dd fe ae
b0| 5e a3 a5 b7 a9 a7 b6 bc bd be 5b 5d af a8 b4 d7
c0| 7b  42 43 44 45 46 47 48 49 ad f4 f6 f2 f3 f5
d0| 7d 4a 4b 4c 4d 4e 4f 50 51 52 b9 fb fc f9 fa ff
e0| 5c f7 53 54 55 56 57 58 59 5a b2 d4 d6 d2 d3 d5
f0| 30 31 32 33 34 35 36 37 38 39 b3 db dc d9 da 00
```

Bytes 0-255: Data from Host

Each byte that is received represents a location, that is, a byte offset, in the appropriate table. For example, if the value 0xC1 (the EBCDIC value for the letter A) is received from the host, it is converted to the value in position 0xC1 in the first table; that is, to 0x41 (the ASCII value for the letter A). This is shown in bold in the preceding table.

Similarly, if the letter *Z* (ASCII value 0x5A) is to be transmitted to the host, it is first converted to the value located in position 0x5A in the second table, which is 0xE9 (the EBCDIC value for Z). This is shown in bold in the table below:

```
   | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00| 00 01 02 03 37 2d 2e 2f 16 05 25 0b 0c 0d 0e 0f
10| 10 14 24 04 b6 15 32 26 18 19 00 27 1c 1d 1e 1f
20| 40 5a 7f 7b 5b 6c 50 7d 4d 5d 5c 4e 6b 60 4b 61
30| f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 7a 5e 4c 7e 6e 6f
40| 7c c1 c2 c3 c4 c5 c6 c7 c8 c9 d1 d2 d3 d4 d5 d6
50| d7 d8 d9 e2 e3 e4 e5 e6 e7 e8  ba e0 bb b0 6d
60| 79 81 82 83 84 85 86 87 88 89 91 92 93 94 95 96
70| 97 98 99 a2 a3 a4 a5 a6 a7 a8 a9 c0 4f d0 a1 00
80| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0| 41 aa 4a b1 9f b2 6a b5 bd b4 9a 8a 5f ca af bc
b0| 90 8f ea fa be a0 b6 b3 9d da 9b 8b b7 b8 b9 ab
c0| 64 65 62 66 63 67 9e 68 74 71 72 73 78 75 76 77
d0| ac 69 ed ee eb ef ec bf 80 fd fe fb fc ad 8e 59
e0| 44 45 42 46 43 47 9c 48 54 51 52 53 58 55 56 57
f0| 8c 49 cd ce cb cf cc e1 70 dd de db dc 8d ae df
```

Bytes 256511: Data to Host

Currently, only the first table (bytes 0255) is used by Host Print service. However, the file must be exactly 512 bytes in length, so the final 256 bytes must be present, even if they are set to zero.

See Also
Reference
Character Tables

Troubleshooting

A variety of diagnostic methods and tools are available to troubleshoot Microsoft Host Integration Server. These methods and tools can provide information about the status of services, connections, 3270, APPC and LUA sessions, and performance. Data can be obtained from the SNA Manager, NetView, RTM, System Monitor, and trace files.

In This Section

Troubleshooting Tools and Tips

Using the Host Integration Server management console, SNA Manager, you can view the status and properties of services, connections, 3270 sessions, APPC sessions, LUA sessions and the number of active users. Server status information includes Active, Inactive, Pending, Stopping, Active [Out of Date] or Error. If the server is active, the number of licensed users and licensed sessions is also shown

If you have an Active [Out of Date] status, the server needs to be restarted to bring the internal parameters up to date with the latest configuration changes. The Error status indicates that an unexpected condition has made the server inaccessible to the SNA Manager.

Connection status can be Active, Pending, Stopping, or Inactive. On Demand connections can also show a status of "On Demand", meaning that the connection will become active when needed. Incoming connections can also show "Incoming", meaning that the connection is available to receive incoming calls.

The status of a non-APPC LU can be Inactive, In Session, SSCP, Available, or Pending. A downstream LU can also have a status of Unavailable. The SSCP status indicates that the LU is in use but is not yet bound to a specific host application. The Available status indicates that the LU is recognized by the host as an available LU. Pending status indicates that a user is trying to access the LU but either the connection is inactive or the mainframe does not recognize the LU.

NetView is helpful for reporting system alerts between mainframes and Host Integration Server. For more information about NetView, see Network Management Support NetView uses NVAlert to provide the ability for the host system console to receive and display alerts generated by Windows 2000 and applications running on the Windows 2000 computer. NVRunCmd provides the ability to command Host Integration Server from the host system console.

Response Time Monitor (RTM) measures the amount of time it takes for a host to respond during 3270 display sessions. RTM is only supported by certain emulators. You can specify the times at which RTM should send data, as well as the trigger that will cause RTM to register that the host has responded. For more information, see Monitoring Mainframe Response Times.

When a problem occurs on Host Integration Server, the Windows 2000 event logs can tell you the sequence and type of events that led up to the problem. For more information about event logs, see Status and Performance Tools.

System Monitor allows you to measure the performance of your computer or other computers on the network. You can monitor connections, LU sessions, and adapters with System Monitor. For more information, see Status and Performance Tools.

Tracing is the action of recording activity between or within components of Host Integration Server. Tracing can provide detailed information of internal activities on the Server. It is helpful in isolating problems and is frequently used by product support personnel.

Four Most Common Problems

The four most common problems reported are as follows:

1. **Setup or Upgrades**: The most frequently asked Host Integration Server questions are related to setup or upgrade issues. If you are connecting to an AS/400 via Token-Ring or Ethernet, make sure you read KB article Q112158 found at http://go.microsoft.com/fwlink/?LinkId=14394. If you are connecting to an AS/400 via an SDLC card, KB Article Q112159 found at http://go.microsoft.com/fwlink/?LinkId=14395 can be helpful. Examples of VTAM configurations for connecting to a mainframe can be found in the Host Configuration section. The configuration file (COM.CFG) contains almost all of the configuration information for the Host Integration Server environment. It is extremely important to back up this file before making changes.

2. **Connectivity Issues**: If you are having a problem connecting to the host, determine where the problem originates. Because Host Integration Server is the middle piece of a three-part connection, it helps to narrow the problem down to a Host Integration client, Host Integration Server issue, or a Host Integration Server host issue.

3. **Host Printing**: For more information, see Host Print Service Problems.

4. **APPC or LUA Application Failures**: See APPC or LUA Application Failures.

APPC or LUA Application Failures

These applications may be written internally, or may be commercial applications. If an error code is produced on an APPC application, it is documented in the "APPC Applications."

An LUA application may produce error codes which are found in the "LUA Applications," or the IBM SNA Formats Guide.

If you are able to test a 3270 or 5250 connection, that can help you determine whether you are experiencing a connectivity problem or a problem with the application itself.

Adapters and Link Service Problems

The following three adapters may have problems associated with installation and configuration. Some of the common problems and solutions are identified in the respective sections.

In This Section

802.2 Adapters and Link Services: Installation Pointers

SDLC Adapters and Link Services: Installation Pointers

X.25 Adapters and Link Services: Installation Pointers

802.2 Adapters and Link Services: Installation Pointers

An 802.2 link service enables the Host Integration Server software to communicate with an Ethernet, Token-Ring, or FDDI adapter. The following pointer highlights an important step for installing an 802.2 adapter and link service:

When you install a new adapter, check the interrupt, port address, and direct memory access (DMA) settings, to avoid conflicts with other adapters in the computer. The normal method for adjusting the settings among your adapters is through EISA configuration programs, jumpers on the motherboard, or configuration utilities. Windows 2000 provides the Device Manager. You may also use configuration utilities supplied with your adapters.

One way of viewing the settings used by various adapters in your computer is to start WINMSD.EXE, the Windows 2000 Diagnostics utility, which includes an IRQ/Port Status button and a DMA/Memory button.

Note
Device drivers provided by Host Integration Server report any interrupt, port address, or DMA conflicts they detect in the Windows 2000 System Event Log. The events are listed under a Source of Service Control Manager or a driver name.

SDLC Adapters and Link Services: Installation Pointers

An SDLC link service enables the Host Integration Server software to communicate with the SDLC adapter. For detailed information about using a particular SDLC adapter or a particular modem (or DCE), see the adapter, modem, or DCE documentation.

The following pointers highlight important steps for installing an SDLC adapter and link service:

- Selecting the Constant RTS option in the **IBM SDLC Link Service Properties** dialog box equates to the NCP LINE macro:

  ```
  DUPLEX=FULL
  ```

  This configuration is recommended for all SDLC connections except tributary connections on multidrop SDLC lines. This configuration improves performance by eliminating several seconds of line turnaround delay after each transmission.

  The Constant RTS option is mandatory if the line is used for full duplex, but it can also enhance performance on a half-duplex line.

- When choosing an SDLC adapter, pay close attention to the speed and duplexing capabilities. Greater speed and/or full-duplexing capabilities allow an adapter to carry greater loads. Note that an adapter that lacks a coprocessor cannot handle

  full-duplex transmission at high transmission speeds, that is, greater than 9600 baud. (Examples of adapters lacking a coprocessor are the IBM SDLC, IBM MPCA, and MicroGate adapters. For these adapters, half-duplex transmission is recommended.)

- Configuring adapters that lack a coprocessor to half duplexing, although slower, has the advantage of requiring substantially less central processing unit (CPU) time. The device driver via DMA will process transmitted information as frames. In contrast, if you use full duplexing with such adapters, the device driver must handle interrupts as characters, placing more load on the CPU. However, if the CPU can handle the load, full duplex provides substantially faster throughput, if the host VTAM configuration also specifies full duplex (DATMODE=FULL in the PU definition).

- After installation of the adapter and link service, you will need to configure the connection with the correct duplex setting. For details, see the next section, X.25 Adapters and Link Services: Installation Pointers.

- Stop all existing services that use the SDLC adapter before attempting to configure the link service properties. For example, stop the Remote Access Service if it uses the SDLC adapter. Only when these services are stopped can SNA Manager successfully autodetect the adapter.

- If you attempt to install an SDLC link service for an adapter that you know is physically installed in the computer, and a pop-up message appears saying that SNA Manager cannot detect the adapter, click **Cancel**. The message indicates that a service is currently using the SDLC adapter. Find out which service it is, stop the service, and then try to install the link service again.

  SNA Manager can detect the adapter only if no other service is using that adapter. If you do not click **Cancel** when the pop-up message appears, SNA Manager continues the installation process using default values (not values attuned to that adapter, since SNA Manager cannot detect it). As described in the preceding paragraph, to correct this situation, stop the service using the SDLC adapter and then reinstall the link service.

  When you install a new adapter, check the interrupt, port address, and direct memory access (DMA) settings, to avoid conflicts with other adapters in the computer. The normal method for adjusting the settings among your adapters is through EISA configuration programs, jumpers on the motherboard, or configuration utilities. Windows 2000 provides the Device Manager. You may also use configuration utilities supplied with your adapters.

  One way of viewing the settings used by various adapters in your computer is to start WINMSD.EXE, the Windows 2000 Diagnostics utility, which includes an IRQ/Port Status button and a DMA/Memory button.

- Device drivers provided by Host Integration Server report any interrupt, port address, or DMA conflicts they detect to the Windows 2000 System Event Log. The events are listed under a Source of Service Control Manager or a driver name.

- To ensure that the line speed is set correctly through hardware or software settings, study the documentation for the modem or DCE. (Do not confuse setting the line speed with selecting the Data Rate, configurable through the **SDLC Connection Properties** dialog box, on the **SDLC** tab, in SNA Manager. The Data Rate setting controls the speed of transmissions between the SDLC adapter and the modem only.)

- If you want to use a server-stored number, check that both the link service and the modem are configured correctly. This applies to MicroGate adapters, or any adapter that can handle a server-stored number. On the modem, it may be necessary to override the setting that causes the modem to autodial from its own stored number. (In modem terminology, when Data Terminal Ready — DTR — is raised, the modem must wait, not dial.) To find out how to override modem autodialing, see your modem or DCE documentation.

X.25 Adapters and Link Services: Installation Pointers

An X.25 link service enables the Host Integration Server software to communicate with the X.25 adapter. The following pointers highlight important steps for installing an X.25 adapter and link service:

- When choosing an X.25 adapter, pay close attention to the speed capabilities. Greater speed allows an adapter to carry greater loads. Note that an adapter that lacks a coprocessor cannot handle high-speed transmission with full duplexing; the duplexing type used by Host Integration Server X.25 connections. (Examples of adapters lacking a coprocessor are the IBM SDLC and IBM MPCA adapters. For these adapters, the transmission speed must be 9600 baud or less.)

- Contact your X.25 carrier or network administrator for information needed for your X.25 link service. The following table lists some important parameters you will need to know:

Parameters from X.25 carrier	**Corresponding parameter in Connection Properties**	**Typical value(s)**
Local X.25 address	Local NUA Address	Varies
Level 3 window size	Default L3 Window Size	2
Level 3 packet size	Default L3 Packet Size	512
Level 2 window size	L2 Window Size	7
Channel ranges (Outgoing SVC, Two-Way SVC, Incoming SVC, and/or PVC)	Channel Ranges (Outgoing SVC, Two-Way SVC, Incoming SVC, and/or PVC)	Vary

  Note that the encoding setting (NRZ or NRZI) is almost always NRZ on X.25 networks. This setting must match the equivalent setting on the host. For connections to mainframes, the NRZI setting is found in the LINE/GROUP definition in VTAM or NCP. If VTAM does not specify the NRZI setting, it defaults to NRZI=YES. For connections to AS/400 computers, the NRZI setting is in the Line Description.

- When you install a new adapter, check the interrupt, port address, and direct memory access (DMA) settings, to avoid conflicts with other adapters in the computer. The normal method for adjusting the settings among your adapters is through EISA configuration programs, jumpers on the motherboard, or configuration utilities. Windows 2000 provides the Device Manager. You may also use configuration utilities supplied with your adapters.

  One way of viewing the settings used by various adapters in your computer is to start WINMSD.EXE, the Windows 2000 Diagnostics utility, which includes an IRQ/Port Status button and a DMA/Memory button.

- Device drivers provided by Host Integration Server report any interrupt, port address, or DMA conflicts that they detect to the Windows 2000 System Event Log. The events are listed under a Source of Service Control Manager or a driver name.

- Ignore the option labeled "Switched: Server-Stored Number" in the **X.25 Link Service Properties** dialog box. This option does not match capabilities currently available with X.25 link services. If chosen, the option behaves the same way as "Switched: Modem-Stored Number."

- With X.25 link services, you can also adjust the T1 timeout and N2 retry limit. However, the defaults for these are appropriate for most networks. Therefore, it is recommended that you do not adjust them unless you have a specific reason for doing so, and that you understand the characteristics of your network as well as the way that the timeout works.

Connection Problems

In This Section

Client to Host Integration Server Problems

When you are troubleshooting an issue between the workstation and the Host Integration Server server, the first step is to verify that the client workstation computer can connect to other network resources on the Windows 2000 Server that has Host Integration Server installed. If you cannot map a network drive at the workstation, then you should troubleshoot this problem as a workstation to Windows 2000 Server issue.

If you can map a network drive at the workstation but cannot get an emulation session, you should determine what protocol is being used. Host Integration Server supports client connections over any of the network protocols. In addition, IPX and TCP/IP connections may be either named pipes or sockets. If you are unsure, run Host Integration Server Setup or Configuration at the client. If TCP/IP or IPX/SPX has been selected, the connection is sockets-based.

If the protocol chosen is "Microsoft Networking," Named Pipes is being used. Depending on which protocols are installed, "Microsoft Networking" will use IPX/SPX, NetBEUI, or TCP/IP.

A common problem preventing workstations from connecting to the Host Integration Server over TCP/IP is failed NetBIOS name resolution. If you can ping the TCP/IP address of the server, but not its NetBIOS name, then you are probably having trouble with WINS or an LMHOSTS file. Configure the client for the IP address rather than the NetBIOS name.

IPX/SPX clients must be able to see the SNA Manager service registered in the Novell Bindery. If the Novell server is version 4.x, Bindery emulation must be enabled except when using the feature in Host Integration Server that supports name resolution via NDS under Netware 4.x. The SNA Manager service registers itself through SAP 444. It is important to verify a router on the network is not filtering this SAP. It is recommended that client workstations be configured for sockets if you are using TCP/IP or IPX/SPX. A configuration of "Remote" causes the client to connect directly to the server. "Local" connects via a broadcast method.

If TCP/IP is being used, the IP address should be entered in the "Primary Server" field rather than the NetBIOS name.

5250 emulation utilizes LU type 6.2 on the AS/400. Often, an error message on the client will include return codes. These codes are documented in the Host Integration Server Message Database included on the CD-ROM.

When using a third-party emulator, the error messages returned by the application will vary. If possible, you should reproduce the error with the client emulator provided with the Host Integration Server and Client software. This will help you verify whether the problem is in the emulator, as well as providing an error message that will probably be documented.

Host Integration Server to Host Problems

Host Integration Server cannot connect to a host over the NetBEUI or IPX/SPX network protocols. DLC, SDLC, and increasingly TCP/IP are the protocols most commonly used to connect from the server to a host. If you cannot get an emulation session at the server itself, check the status of the connection using the SNA Manager. If the status is "Active", but you cannot access an emulation session, there may be a configuration problem. If the connection to an AS/400 is "Pending", it may be useful to go through the KB articles Q112158 found at: http://go.microsoft.com/fwlink/?LinkId=14394 or Q112159 found at: http://go.microsoft.com/fwlink/?LinkId=14395 or browse other KB articles available from the Web site http://go.microsoft.com/fwlink/?LinkId=14396 to make sure that you have all the required information.

If the status is "Inactive" or "Pending", the communication between the server and the host is failing.

Check the Application and System Event Logs. Most often, "Pending" or "Inactive" connections will produce an event in the Applications Log – usually an event 230 or 23. The text of these messages varies and can indicate the nature of the problem. If the problem is the result of hardware failure, the event will usually be logged in the System Log. If the connection has been functional in the past, there may be a communications problem. For example, an SDLC connection may fail due to phone line problems.

Settings to Check for All Connection Types

When you are configuring a new Host Integration Server connection or troubleshooting an existing connection, regardless of the connection type, the identifiers must match between the Host Integration Server computer and the host. The type of identifier (ID or name) is as follows:

For most mainframe connections

Node ID is the identifier used when exchanging identification (XIDs) with most mainframes. Check to make sure that the following items match; if they do not, the Host Integration Server computer is not identifying itself in a way that the host can recognize.

Identifier used on mainframe	Identifier to configure on Host Integration Server
IDBLK and **IDNUM** in the PU definition	Two parts of the **Local Node ID** (configured on the connection)

For other mainframe connections

There are some situations in which the mainframe does not use Node ID in XIDs, but instead uses Network Name and Control Point Name. These situations include mainframes communicating through LU 6.2, and mainframes that call up the Host Integration Server computer (meaning that the Host Integration Server computer accepts incoming calls on that mainframe connection). In these situations, the following parameters must match.

Note
Change the following identifiers only when necessary. The local Network Name and the local Control Point Name should be changed only when the host requires a specific Network Name and Control Point Name that differ from those configured in t he server properties.

Identifier used on mainframe (in unusual cases only)	Identifier to configure on Host Integration Server
NETID in the VTAM Start command for the local SSCP **CPNAME** in the PU definition**NETID** and **SSCPNAME** in the VTAM Sta rt command for the remote SSCP (VTAM system)	**Local Network Name** (configured on the server) **Local Control Point Name** (configured on the server) **Remote Network Name** and **Remote Control Point Name** (configured on the connection)

For AS/400 connections

Network Name and Control Point Name (used together and called the fully qualified name) are the identifiers used when exchanging identification (XIDs) with AS/400 computers. Check to make sure that the following items match. If they do not, the Host Integration Server is not identifying itself in a way that the AS/400 can recognize.

Note
Change the following identifiers only when necessary. To support multiple connections to the same AS/400, you must provid e a unique local Network Name and local Control Point Name for each connection. In addition, you can change the local Net work Name and the local Control Point Name if the host requires a specific Network Name and Control Point Name that diff er from those configured in the server properties.

Identifier used on AS/400	Identifier to configure on Host Integration Server
RMTNETID (usually; often set to APPN); **RMTCPNAME**	**Local Network Name** and **Local Control Point Name** (confi gured on the server)
RMTNETID (often set to APPN); **CP Name** (shown in the "Disp lay network attributes" screen)	**Remote Network Name** and **Remote Control Point Name** (configured on the connection)

802.2 Connection Pointers

Connections are configured through Host Integration Server SNA Manager. The steps for configuring an 802.2 connection are described in detail in Important Connection Information. The following pointers indicate items that require special attention:

- As with all connections, the identifiers must be configured correctly. For details, see Settings to Check for All Connection Types.

- Pay close attention to the Remote Network Address (which can be viewed on the **Address** tab of **Connection Properties**). It should match the 12-digit hexadecimal network address of the remote host, peer, or downstream system. The following guidelines may help:

For...	Set the Remote Network Address to...
Connections to a 3174 controller	The value in the configuration response 900 of the controller's customization program.
Connections to a 3720, 3725, or 3745 front-end processor	LOCADDR in the NCP LINE macro.
Connections to an IBM mainframe	MACADDR in the VTAM PORT definition
Connections to an AS/400	ADPTADR in the Line Description on the AS/400

- Check the Max BTU Length setting, found on the **DLC 802.2** tab of **Connection Properties**. A BTU is also called an I-frame. It is the number of bytes that can be transmitted in a single data-link information frame.

  Set the Max BTU Length so that it matches the capacity of the adapter and the host or downstream system. Otherwise, when the mainframe or AS/400 sends a logon screen, the BTU length (frame size) that is used will be unworkable, causing the connection to be dropped. The following table provides guidelines:

Adapter type	Max BTU Length to use
Token-Ring adapter that transmits at 4 Mbps (megabits per second)	Less than or equal to 4195.
Token-Ring adapter that transmits at 16 Mbps	The appropriate value for the host or downstream system; the adapter itself can handle the greatest Max BTU Length available with Host Integration Server, 16393.
Ethernet adapter	Less than or equal to 1493.

Connection type	Max BTU Length to use
Connection to a mainframe	Equal to MAXDATA in the PU definition on the mainframe (recommended).
Connection to an AS/400	Equal to MAXFRAME on the AS/400 (recommended).
Connection to a downstream system	Less than or equal to the maximum value supported by the downstream system.

Setting the Max BTU Length correctly is especially important for downstream connections.

With 802.2 connections, you can adjust a variety of other settings, especially timeout and retry settings. However, the defaults for the timeouts and retries are appropriate for most networks. Therefore, it is recommended that you do not adjust these settings unless you have a specific reason for doing so, and unless you understand the characteristics of your network, and the way that the timeouts work.

See Also
Other Resources
Connection Problems

Troubleshooting 802.2 Connections

The following table outlines some ways to interpret symptoms of a nonfunctioning 802.2 connection:

Symptom	Configuration item to check
Symptom: Host Integration Server will not start; the System Event Log (viewable through the Event Viewer) contains messages about a link service failing to start.	**Recommendation**: Check that all existing link services are configured to work with functioning adapters. The server service cannot start if any existing link service cannot start, even if that link service is not being used by any connection.
Symptom: Host Integration Server will not start; System Event Log contains messages about conflicts in port addresses, IRQs, or DMA addresses.	**Recommendation**: Check the port address, IRQ setting, or DMA address of your 802.2 adapter, which may be conflicting with those of another adapter or port in your computer. Change the settings on one of the adapters or ports, if possible, or remove the conflicting adapter.
Symptom: Connection can only reach "Pending" status, and Application Event Log contains event 230.	**Recommendation**: Check that the identifiers (IDs or names) and/or the network address used by the host match the corresponding parameters in SNA Manager. (To view these parameters, double-click the connection in the left pane and then click **System Identification**.) The identifiers or addresses must match in order for the exchange of identifiers (XIDs) and test frames to complete successfully. For more information about identifiers, see the preceding section.
Symptom: Connection (especially downstream connection) may become "Active," but reverts to "Pending" when first user attempts to start a session.	**Recommendation**: Check the Max BTU Length setting for the connection (to see the setting, in SNA Manager, double-click the connection in the left pane and then click **DLC 802.2**). For details about the correct setting to use, see the preceding section and Settings to Check for All Connection Types.
Symptom: Connection reaches "Active" status, but reverts to "Pending;" event 23 is generated in the Application Event Log.	**Recommendation**: Possible causes include LAN or router problems, or actions by the remote system (for example, an AS/400 disconnecting because of lack of activity).

Note

With 3270 LUs, if one or more of the LUs never become active even when users are attempting to start sessions, the LOCADDR settings in the LU definitions in VTAM or NCP may not match the Host Integration Server LU numbers, or some or all of the LUs may be inactivated in VTAM or NCP. To correct this, consult with the administrator of the host system.

SDLC Connection Pointers

Connections are configured through the SNA Manager. The steps for configuring an SDLC connection are described in detail in Important Connection Information. The following pointers indicate items that require special attention:

As with all connections, the settings must be configured correctly. For details, see Settings to Check for All Connection Types.

If possible, configure the connection to use constant RTS. See the recommendations in SDLC Adapters and Link Services: Installation Pointers.

The duplex setting is found in the **SDLC** tab of **SDLC Connection Properties**. Duplex can be half or full; the setting must not exceed the capabilities of the adapter and modem. For more information about the factors to consider when choosing the setting for duplex, see the preceding section.

Check the encoding setting, which is either nonreturn to zero (NRZ) or nonreturn to zero inverted (NRZI). The setting is on the **Address** tab of **SDLC Connection Properties**. This setting must match the equivalent setting on the host. For connections to mainframes, the NRZI setting is found in the LINE/GROUP definition in VTAM or NCP. (If VTAM does not specify the NRZI setting, it defaults to NRZI=YES.). For connections to AS/400 computers, the NRZI setting is in the Line Description.

When multiple SDLC connections accept incoming calls and use the same link service, the encoding (NRZ/NRZI) settings for all the connections must match.

Check the Max BTU Length setting, found on the **SDLC** tab of the **SDLC Connection Properties** dialog box. A BTU is also called an I-frame; it is the number of bytes that can be transmitted in a single data-link information frame.

Set the Max BTU Length so that it matches the capacity of the adapter and the host or downstream system. Otherwise, when the mainframe or AS/400 sends a logon screen (typically a full screen), the BTU length (frame size) that is used will be unworkable, causing the connection to be dropped. The following table provides guidelines:

For...	**Set the Max BTU Length to...**
IBM SDLC adapter	521 or less.
Connection to a mainframe	Equal to MAXDATA in the PU definition on the mainframe (recommended).
Connection to an AS/400	Equal to MAXFRAME on the AS/400 (recommended).
Connection to a downstream system	Less than or equal to the maximum value supported by the downstream system.

Setting the Max BTU Length correctly is especially important for downstream connections.

With SDLC connections, you can adjust a variety of other settings, especially timeout and retry settings. However, the defaults for the time-outs and retries are appropriate for most networks. Therefore, it is recommended that you do not adjust these settings unless you have a specific reason for doing so, and you understand the characteristics of your communications lines as well as the way that the time-outs work.

Troubleshooting SDLC Connections

To begin troubleshooting an SDLC connection, it is helpful to look at two simple indicators of activity: the server and connection status shown in SNA Manager, and the modem lights. Event logs also provide important information. The following table outlines some ways to interpret symptoms of a nonfunctioning SDLC connection.

Symptom	Configuration item to check
Symptom: Host Integration Server will not start; the System Event Log (viewable through the Event Viewer) contains messages about a link service failing to start.	**Recommendation**: Check that all existing link services are configured to work with functioning adapters. The SNA Manager service cannot start if any existing link service cannot start, even if that link service is not being used by any connection.
Symptom: Host Integration Server will not start; System Event Log contains messages about conflicts in port addresses, IRQs, or DMA addresses.	**Recommendation**: Check the port address, IRQ setting, or DMA channel of the SDLC adapter, which may be conflicting with those of another adapter (for example, a sound system adapter) or port in your computer. Change the settings on one of the adapters or ports, if possible, or remove the conflicting adapter. For more information, see SDLC Adapters and Link Services: Installation Pointers.
Symptom: Modem lights may flash but the connection remains at "Pending" and does not go to "Active."	**Recommendation**: Check the DMA channel setting (jumpers) on the adapter and compare this to the setting in the link service properties dialog box. If this setting specifies a nonexistent address, data cannot flow.
Symptom: Modem appears to make the connection ("receive data" light flashes, but "transmit data" light does not). The connection status reaches "Pending" and then the connection is dropped.	**Recommendation**: Check the cables; then check the Encoding (NRZ/NRZI) setting used by the host and compare this to the setting in the connection properties dialog box. If the NRZ/NRZI settings do not match, the two ends of the connection begin negotiating but cannot interpret each others signals correctly. For more information, see the preceding section.
Symptom: Modem lights are flashing but the connection can only reach "Pending" status. Also, the Application Event Log contains event 182.	**Recommendation**: Check that the identifiers (IDs or names) used by the host match the ones used in the **Connection Properties** dialog box. If the identifiers do not match, the exchange ID (XID) process cannot complete successfully. For more information about identifiers, see Settings to Check for All Connection Types.
Symptom: Connection (especially downstream connection) may become "Active," but reverts to "Pending" when first user attempts to start a session.	**Recommendation**: Check the Max BTU Length setting in the **Connection Properties** dialog box. For details about the correct setting to use, see the preceding section.

If one or more 3270 LUs never becomes active, even when users are attempting to start sessions, the LOCADDR settings in the LU definitions in VTAM or NCP may not match the Host Integration Server LU numbers, or some or all of the LUs may be inactivated in VTAM or NCP. To correct this, consult with the administrator of the host system.

X.25 Connection Pointers

An X.25 connection uses a packet-switching network, and communicates through the Qualified Logical Link Control (QLLC) protocol. Connections are configured through SNA Manager. The steps for configuring an X.25 connection are described in detail in Important Connection Information. The following pointers indicate items that require special attention:

- As with all connections, the identifiers must be configured correctly. For details, see Settings to Check for All Connection Types.

- Pay close attention to the **Remote X.25 Address** on the **Address** tab of the **Connection Properties** dialog box; it should match the address of the remote host, peer, or downstream system. The address consists of from 1 through 15 hexadecimal digits. If a 15-digit address is used, the final 3 digits are used for routing between stations with the same 12-digit address. For connections to a host using VTAM, the address should match the DIALNO parameter in the PORT definition. For connections to an AS/400, the address should match the local address of the AS/400 (assigned by the X.25 carrier).

- Check the Max BTU Length setting, found in the **Connection Properties** dialog box. A BTU is a data-link information frame.

- Set the Max BTU Length so that it matches the capacity of the adapter and the host or downstream system. Otherwise, when the mainframe or AS/400 sends a logon screen, the BTU length (frame size) that is used will be unworkable, causing the connection to be dropped. The following table provides guidelines:

For...	**Set the Max BTU Length to...**
IBM SDLC adapter	521 or less.
Connection to a mainframe	Equal to MAXDATA in the PU definition on the mainframe (recommended).
Connection to an AS/400	Equal to MAXFRAME on the AS/400 (recommended).
Connection to a downstream system	Less than or equal to the maximum value supported by the downstream system.

- Setting the Max BTU Length correctly is especially important for downstream connections.

- For SVC connections, check that the codes in **User Data** include C3, which specifies the QLLC protocol (the X.25 protocol used by SNA Manager). To view **User Data** codes, right-click the connection in the left pane, then click **Properties**, then click the **X.25** tab in the **Connection Properties** dialog box. For information about other user data codes, contact your X.25 network administrator.

- If each attempt at connection activation incurs cost as you access the X.25 network, you may want to limit the number of these attempts. To do this, double-click the connection, and then click the **X.25** tab. In the **Maximum Retries** list, select an appropriate maximum number of attempts. (The default is No Limit.)

Troubleshooting X.25 Connections

The following table outlines some ways to interpret symptoms of a nonfunctioning X.25 connection.

Symptom	Configuration item to check
Symptom: Host Integration Server will not start; the System Event Log (viewable through the Event Viewer) contains messages about a link service failing to start.	**Recommendation**: Check that all existing link services are configured to work with functioning adapters. The SNA Manager cannot start if any existing link service cannot start, even if that link service is not being used by any connection.
Symptom: Host Integration Server will not start; System Event Log contains messages about conflicts in port addresses, IRQs, or DMA addresses.	**Recommendation**: Check the port address, IRQ setting, or DMA address of your X.25 adapter, which may be conflicting with those of another adapter or port in your computer. Change the settings on one of the adapters or ports, if possible, or remove the conflicting adapter. For more information, see X.25 Adapters and Link Services: Installation Pointers.
Symptom: Connection can only reach "Pending" status, and the Application Event Log contains Event 23 with an outage code of 0x62 (for SVC) or 0x61 (for PVC).	**Recommendation**: Check that the identifiers (IDs or names) and/or the X.25 address used by the host match the corresponding parameters on the **System Identification** tab of the **Connection Properties** dialog box. The identifiers or addresses must match in order for the exchange of identifiers (XIDs) and test frames to complete successfully. For more information about identifiers, see the preceding section and Settings to Check for All Connection Types.
Symptom: Connection (especially downstream connection) may become "Active," but reverts to "Pending" when first user attempts to start a session.	**Recommendation**: Check the **Max BTU Length** setting for the connection on the **X.25** tab of the **Connection Properties** dialog box. For details about the correct setting to use, see the preceding section.

If one or more 3270 LUs never becomes active, even when users are attempting to start sessions, the LOCADDR settings in the LU definitions in VTAM or NCP may not match the Host Integration Server LU numbers, or some or all of the LUs may be inactivated in VTAM or NCP. To correct this, consult with the administrator of the host system.

Settings to Check on Channel Connections

The number of parameters involved in channel connections is fewer than with other types of connections. The key parameters to note are those specifying addresses and, for channel connections, the Max BTU Length.

If one or more 3270 LUs never becomes active, even when users are attempting to start sessions, the LOCADDR settings in the LU definitions in VTAM or NCP may not match the Host Integration Server LU numbers, or some or all of the LUs may be inactivated in VTAM or NCP. To correct this, consult with the administrator of the host system.

Event and Error Problems

In This Section

Additional Help with Events and Errors

Microsoft provides troubleshooting information about specific product event and error messages. For Microsoft Windows XP and Microsoft Windows Server 2003 operating systems, you can access event message documentation by using the link inside the property window of an event. Microsoft also provides this content on the Web at http://go.microsoft.com/fwlink/?LinkId=20869.

The information provided on the Web site includes:

- The text of the event or error message, so you can confirm you have located the correct message
- A detailed explanation of what caused the event or error
- Recommended actions, if any, for you to take to correct the problem.

For Microsoft Windows 2000, Microsoft Windows XP, and Microsoft Windows Server 2003 operating systems, you can also access additional information about an event by using the Event Viewer or additional information about an error directly from the Microsoft website.

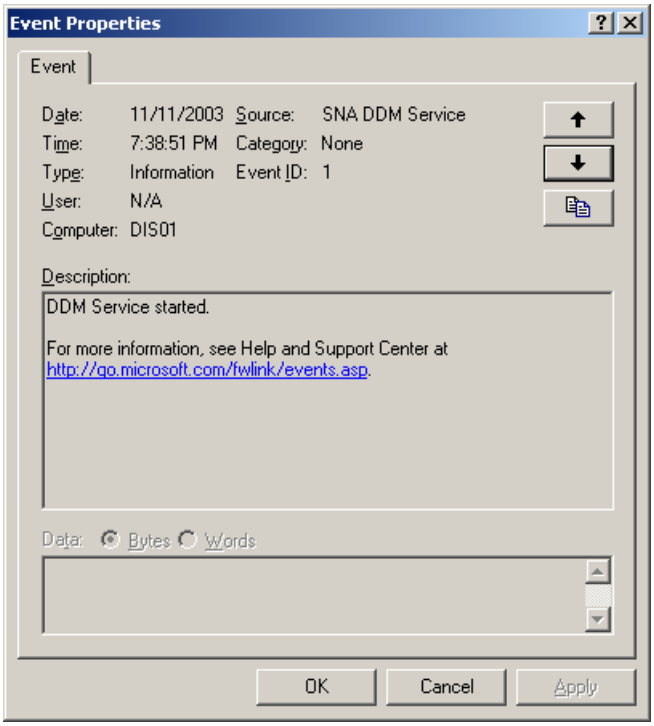
To access additional information about events through the Windows 2000 Event Viewer

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Event Viewer**.
2. Expand the **Event Viewer (Local)** folder. You should see at least three logs: **Application**, **Security**, and **System**.
3. Click the **Application log** to view its list of events.
4. Double-click the event of interest.
5. Under **Description**, look for **For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp**, and then click the link.
6. In the **Event Viewer** dialog box, click **Yes**.
7. When the results appear, select the message text that matches the message you received. Any relevant Knowledge Base articles are also available on the page.

To access additional information about events through the Windows XP/Windows Server 2003 Event Viewer

1. Click Start, point to Programs, point to Administrative Tools, and then click Event Viewer.
2. Expand the Event Viewer (Local) node.
3. Click Application to view a list of events.
4. Double-click an event.
5. Under the Description text box, click http://go.microsoft.com/fwlink/events.asp.

   Event Properties for Event Viewer



6. In the **Event Viewer** dialog box, click **Yes**.

Note
Clicking **Yes** allows your computer to send information across the Internet.

Note
The **Help and Support Center** dialog box displays the event ID event source, detailed explanation, and recommended actions.

Information about errors and events, as well as relevant Knowledge Base articles, are also available directly from the Microsoft Web site at http://go.microsoft.com/fwlink/?LinkId=20869.

Microsoft continually updates the troubleshooting information about specific product event and error messages and posts this information to http://go.microsoft.com/fwlink/?LinkId=20869. The following table lists the event and error ID numbers that had additional information available on the Web at the time Host Integration Server was released. Microsoft recommends that you check the Web site for the additional messages that have been added since product release.

Connection Initialization Sequence

When troubleshooting pending DLC Connections, it is helpful to review the connection initialization sequence that occurs between the Host Integration Server computer and the host platform.

The following diagram applies to both AS/400 and mainframe connections:

Connection initialization sequence

CONNECTION INITIALIZATION: Ethernet 802.2

Connection Initialization Overview

Connection Initialization occurs in a sequential fashion. Each step builds upon the preceding step. After all steps have completed correctly, an Active connection is established between Host Integration Server and a remote host, either an AS/400 or a mainframe.

Every connection requires that a number of parameters match between the two platforms. Throughout this overview, these parameters, indicated in bold type, are displayed using the actual field names used by Host Integration Server and a remote host (that is, upper/lower case is preserved).

Steps 1 and 2

Verify basic network connectivity - can Host Integration Server communicate with the host's network adapter card?

Does the **Remote Network Address** on the **Address** tab of the **Connection Properties** dialog box match the **Local Adapter Address** defined in the AS/400 line description, or the **Switched Line** in the VTAM definition?

Steps 3 and 4

Verify host SSAP - is the host listening on the correct Source Service Access Port?

Does the **Remote SAP Address** on the **Address** tab of the **Connection Properties** dialog box match the **LAN SSAP** defined in the APPC Controller Description, or the **Switched Line** in the VTAM definition?

Steps 5 and 6

Verify local parameters - do the Host Integration Server "Local" parameters on the **System** tab of the **Connection Properties** dialog box match the "Remote" parameters in the host APPC controller, or VTAM PU, definition?

Step 7

Verify remote parameters - do the Host Integration Server "Remote" parameters, also defined on the **System** tab of the **Connection Properties** dialog box, match the "Local" parameters in the host APPC controller, or VTAM PU definition?

Connection initialization is a complex, sequential process that includes a wealth of details.

For more information, see Connection Initialization Details.

Connection Initialization Details

For an overview, refer to the topic Connection Initialization Overview.

Step 1, outbound

When Host Integration Server attempts to activate a DLC connection, it first sends out an LLC TEST frame to the host's network adapter address to initiate the link level connection, and to verify the link station to link station transmission path.

This TEST command is sent by the SNADLC 802.2 link service via the Window's DLC driver.

Host Integration Server issues this TEST command at the start of every 802.2 connection. Although this command does not cause connection activation, its failure precludes continuing the activation.

This TEST command causes the remote link station to return a TEST Response. This will not affect the mode or state of the remote link station.

TOKEN-RING CONNECTIONS:

The TEST command is sent to the local ring first.

If Host Integration Server does not receive a reply to this TEST command within 0.5 seconds, it resends it, with the "all routes broadcast" setting enabled, causing it to be forwarded to adjacent rings by any source routing bridges which are present on the ring. The SNADLC link service will send a total of three all-routes broadcast TEST commands if the remote station (host) is not responding.

ETHERNET CONNECTIONS:

The TEST command does not contain any source-routing information.

Host Integration Server sends four TEST commands to the host's network adapter address.

1. 802.3 to the regular MAC address
2. 802.3 to the bit flipped MAC address
3. DIX format to the regular MAC address
4. DIX to the bit flipped MAC address

Step 2, inbound

The success of the TEST command indicates that the host's network interface adapter is accessible, and is configured for SNA communication.

The failure to receive a response to the TEST command will result in an Event 230 "No response to TEST commands".

What does this failure imply?

1. The **Remote Network Address**, which is the host's network adapter address, is incorrectly configured on the Host Integration Server Connection Properties dialog box, Address Tab. It should match the **Local Adapter Address** defined in the AS/400 line description, or the **Switched Line** in the VTAM definition.
2. Multiple Network Adapter Cards: If the Host Integration Server computer has multiple network adapter cards, say one for the SNA Link Service, and the other for Windows 2000/NT domain connectivity, and the SNA Link Service is configured to communicate over the NT domain adapter, there obviously will be no AS/400 response.
3. Intermediate bridges or routers are not passing the DLC Protocol between the Host Integration Server computer and the host. DLC traffic cannot reach the segment or ring on which the AS/400's network address resides because the intermediate locater is not forwarding the LLC frames.
4. The host is not enabled for SNA communication (this occurs rarely).

Step 3, outbound

Verify that the host's **LAN SSAP** matches the Host Integration Server **Remote SAP Address** property.

A NULL XID command is sent to the host. This is required as part of link level connection establishment. This command is sent to the host's **LAN SSAP** (which is referred to in the Host Integration Server environment as the **Remote SAP Address**). This is the port that the host 'listens to', when receiving messages from the Host Integration Server computer. SNA protocol uses a default **LAN SSAP** of 0x04.

Step 4, inbound

The success of the NULL XID command is indicated when Host Integration Server receives back a RQOS (Request Open Station) XID response.

The failure to receive a RQOS command will also result in the generation of an Event 230 – "No Response to XID Commands".

This indicates that the Host Integration Server timed out before the RQOS command was received, because:

- The Host Integration Server **Remote SAP Address** was incorrect.
- The AS/400 was unable to create a new APPC Controller for some reason. For example, AUTOCRTCTL is disabled.
- The AS/400 APPC Controller or VTAM PU is in an Error State, or Inactive Status.

On Ethernet and Token-Ring networks, if the host does not reply to the XID command, then the connection remains in a pending condition, and an Event 230 will be logged in the NT Event Viewer Application log. The Host Integration Server computer will continually attempt connection startup, though only the first Event ID 230 will be logged with the Event Viewer, to prevent the log from filling up.

Step 5, outbound

Verify that the Host Integration Server computer's "Local Parameters" match an APPC Controller, or VTAM PU, description on the host.

An XID command is sent to the host. This XID contains the Host Integration Server computer's "Local Parameters". The parameters of interest are the Host Integration Server computer's network adapter address, "Local" **Network Name**, and "Local" **Control Point Name**.

Step 6, inbound

The success of the "Local Parameters" XID is indicated when Host Integration Server receives back a "Remote Parameters" XID from the host containing its "Local Parameters".

The failure to receive this "Remote Parameters" XID from the host will result in the generation of an Event 56 - "XID rejected by remote computer

The host failed to match the parameters sent in Step 5 to a known APPC controller or VTAM PU definition.

Host Integration Server logs Event 56 when it receives an XID Negotiation Error (X'22') Control Vector from the host. This control vector includes an offset pointer into the XID that was sent to the host in Step 5, where the offset points to the parameter in error. If the offset is "2" (pointing to the Local Node ID parameter), then an Event 47 is logged. If the offset is 19 (pointing to the link role parameter), then an Event 46 is logged. These two events are very rare. If some other offset is received (99.9% of the time), then Event 56 is logged.

For the purpose of discussion, lets assume an APPC Controller already exists on the host containing:

- The address of the network adapter in the Host Integration Server computer, which is referred to on the AS/400 as the **LAN remote adapter address** (ADPTADR).
- A specific **Remote network identifier** (for example, APPN).
- A specific **Remote control point** (for example, TEST1).

The XID sent in step 5 will be rejected under the following circumstances:

1. There is already an APPC controller defined on the AS/400 with the same name as the name defined in Host Integration

Server SNA Manager, but it has a different **LAN remote adapter address** (ADPTADR)" associated with it.

Another way of saying the same thing is: The Host Integration Server computer attempted to connect with an invalid **"Local" Network Name**. For example: ESSLAB, where the AS/400 was expecting APPN.

(There is a source of potential confusion here. The **Remote Network Identifier** in the APPC Controller description corresponds to the **"Local" Network Name** on the Host Integration Server **Connection Properties** dialog box, **System** tab. However, on the AS/400 Display Networks Attributes screen, you will very likely see that the value for the **Local network ID** is identical to the APPC **Remote Network Identifier**. The confusion exists because the Host Integration Server **"Local" Network Name** is apparently referred to by two different field names on the AS/400. However, these two AS/400 fields *are not* the same. The AS/400 **Remote Network Identifier** *must* match the Host Integration Server **"Local" Network Name.** The AS/400 **Local network ID** does not have to match. It is usually defaulted to the same name as the **Remote Network Identifier** for the sake of convenience, however, the name actually refers to the AS/400's "local" network ID, *not* the "local" network name on Host Integration Server).

2. There is already a different controller name defined on the AS/400 with the same "**LAN remote adapter address** (ADPTADR)" (Autocreate Controller (AUTOCRTCTL) is set to YES in the AS/400 Line Description).

   In other words, The Host Integration Server computer attempted to connect with an invalid **"Local" Control Point Name**. For example: TESTXXX, where the AS/400 was expecting TEST1.

3. The **Remote Control Point Name** in the Host Integration Server **Connection Properties** dialog box does not match the **Local Control Point Name** in the AS/400 Network Attributes screen.

4. The Host Integration Server computer is configured with XID Type: Format 0, when the AS/400 requires XID Type: Format 3.

Step 7

Host Integration Server uses a four-step algorithm to match the hosts "Local parameters". If a step succeeds, any remaining steps are bypassed, and the activation sequence continues to step 8.

1. The **Network Name** and **Control Point Name** from the host XID are compared to the **Network Name** and **Control Point Name** in the "Remote" Node Name section of the Host Integration Server **Connection Properties** dialog box, **System Identification** tab.

2. If the host XID does not contain a network name and control point name (format 0 XID) or those fields are blank in the Host Integration Server **Connection Properties** dialog box, the matching will proceed to the next step. If the comparison fails, the XID will be rejected and event ID 49 is generated. If the comparison is successful, the XID will be responded to in step 8.

3. If Host Integration Server is unable to match the **Network Name** and **Control Point Name**, it will compare the IDBLK and IDNUM (Node ID) in the XID to the "Remote" Node ID field in the Host Integration Server **Connection Properties** dialog box. If this field is blank, the matching will proceed to the next step. If the comparison fails the XID is rejected and NT Event ID 49 is generated.

Note
All XIDs must contain an IDBLK and IDNUM.

   If the comparison is successful, the XID will be responded to in step 8.

4. Host Integration Server then compares the network address contained within the incoming XID to the **Remote Network Address** configured in the Host Integration Server **Connection Properties** dialog box. For x.25 the Remote X.25 address is used. If this comparison fails, the XID is rejected and NT Event ID 49 is generated. For SDLC the XID is simply accepted.

If the comparison process fails, an Event 49 is generated by Host Integration Server, and logged with the NT Event Service.

Step 8

The connection eventually goes active.

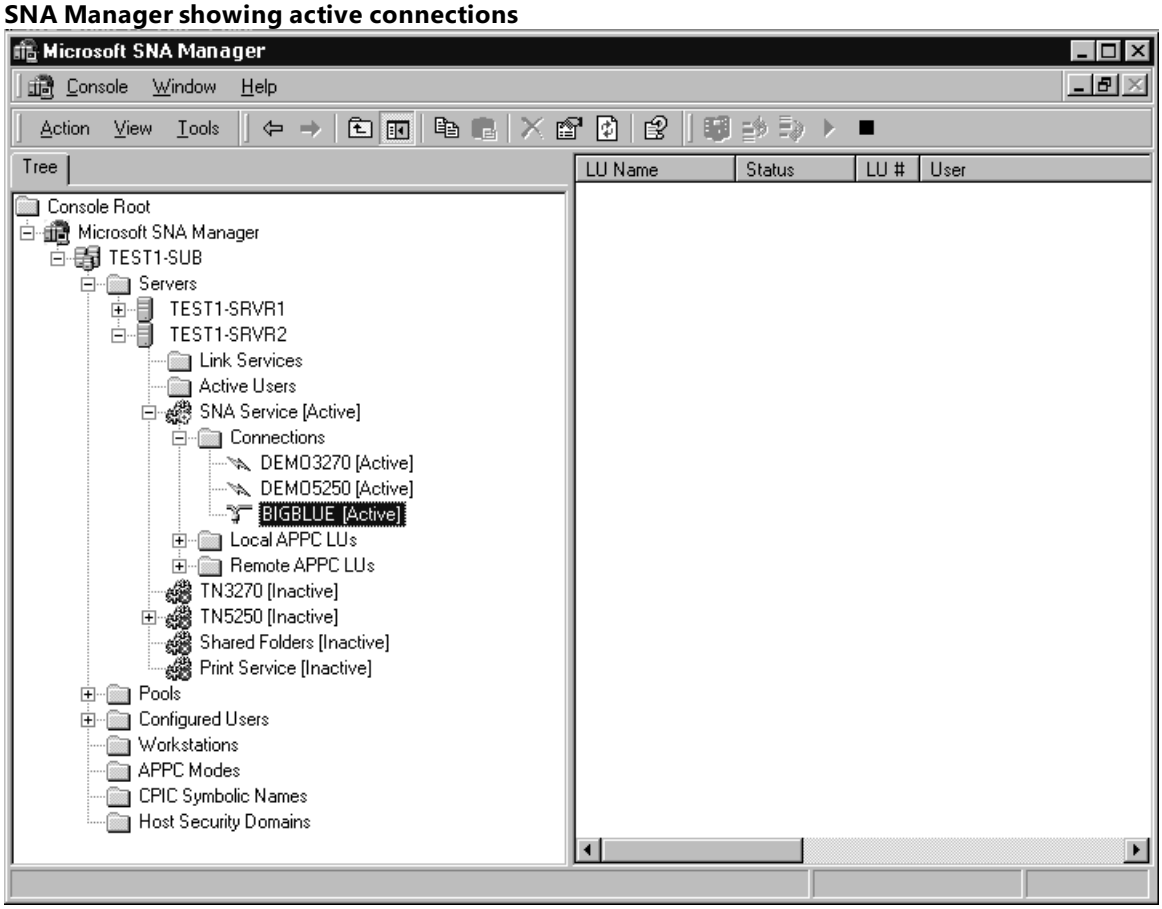
Finding Relevant Information

In This Section

Host Integration Server Screens

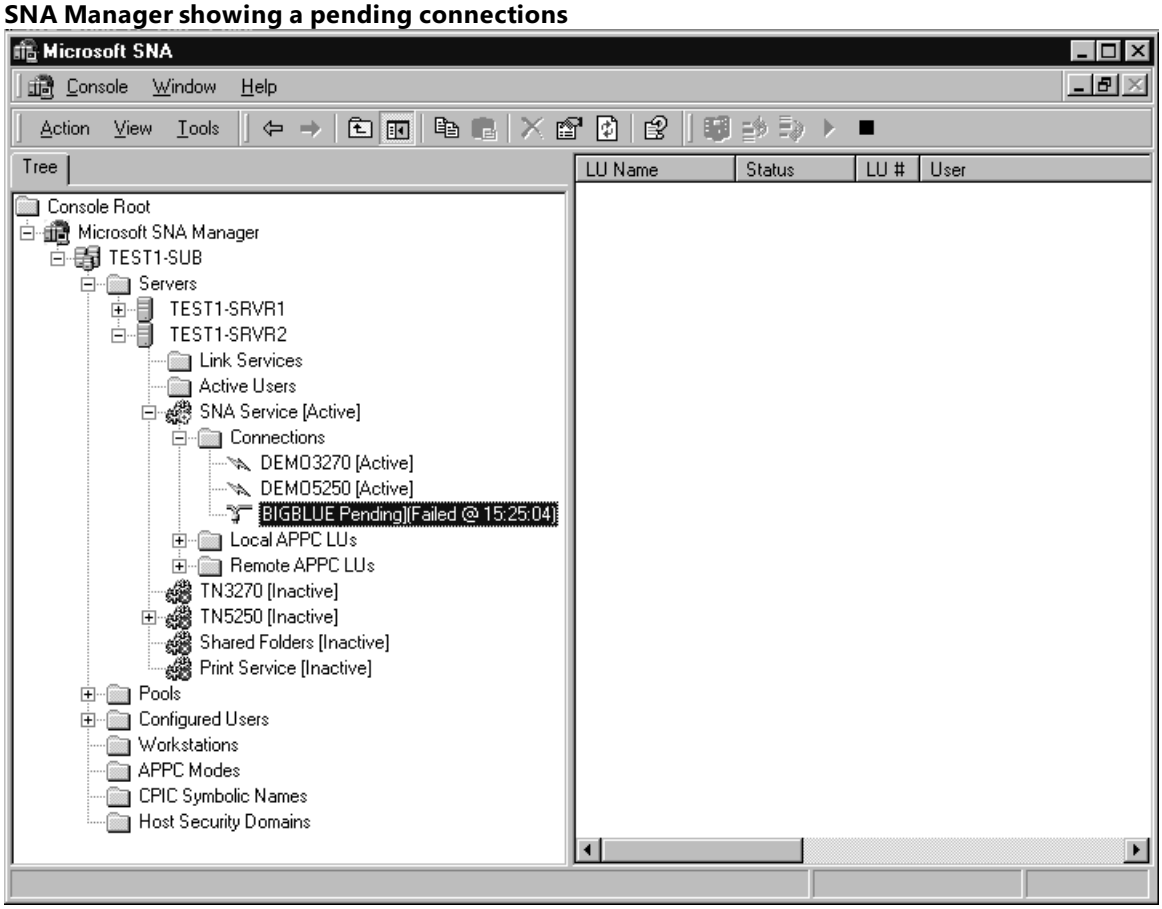
Here is the SNA Manager showing active connections:

SNA Manager showing active connections



Here is the SNA Manager showing a pending connection:

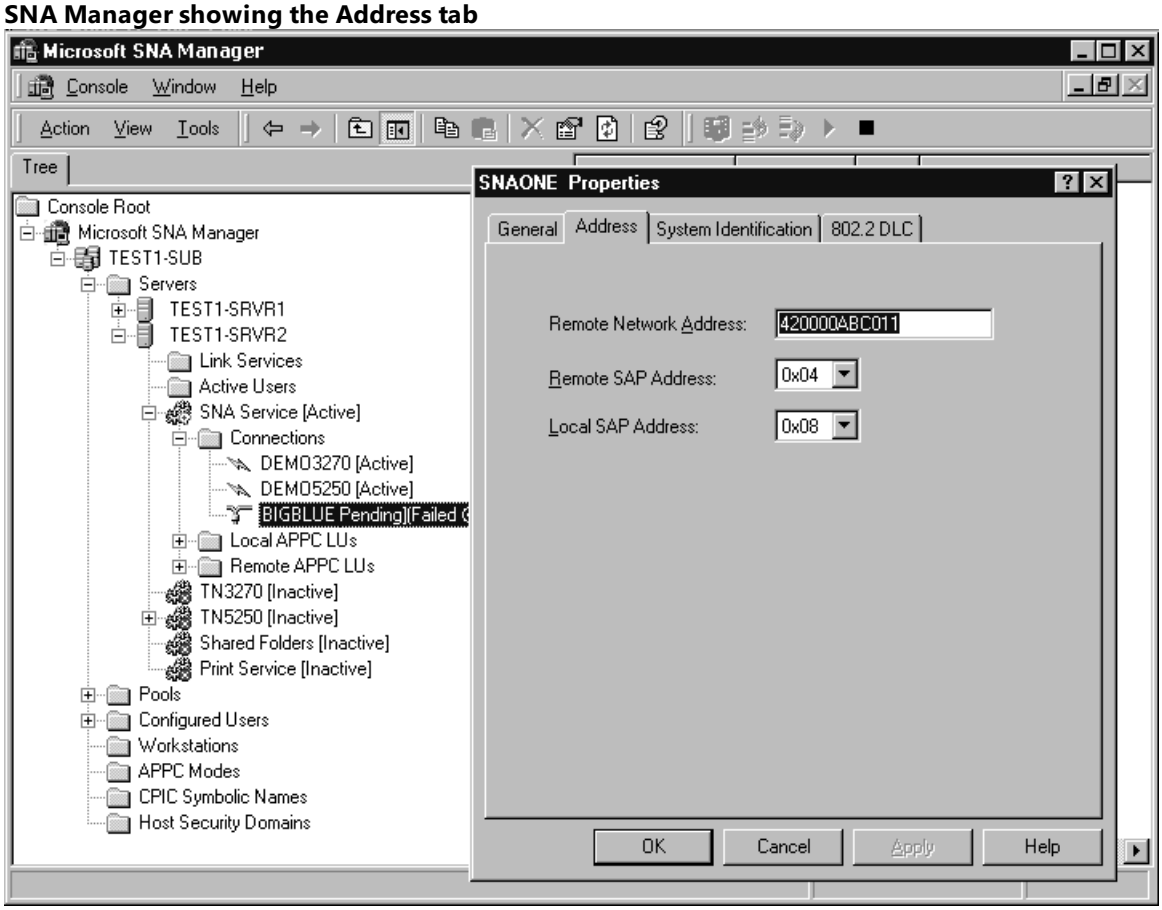
SNA Manager showing a pending connections



Right-click the pending connection, and then click **Properties**.

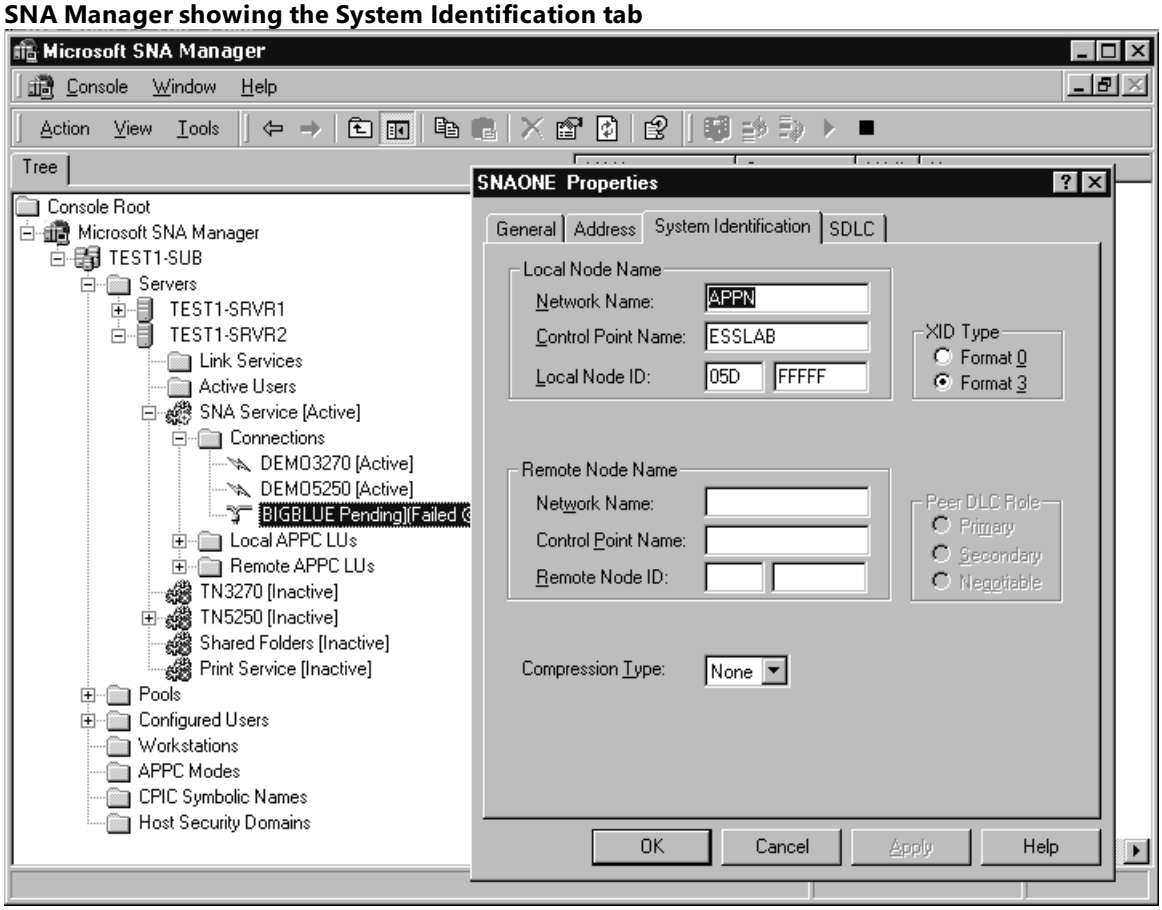
Here is the SNA Manager with the **Address** tab selected:

SNA Manager showing the Address tab



Here is the SNA Manager with the **System Identification** tab selected.

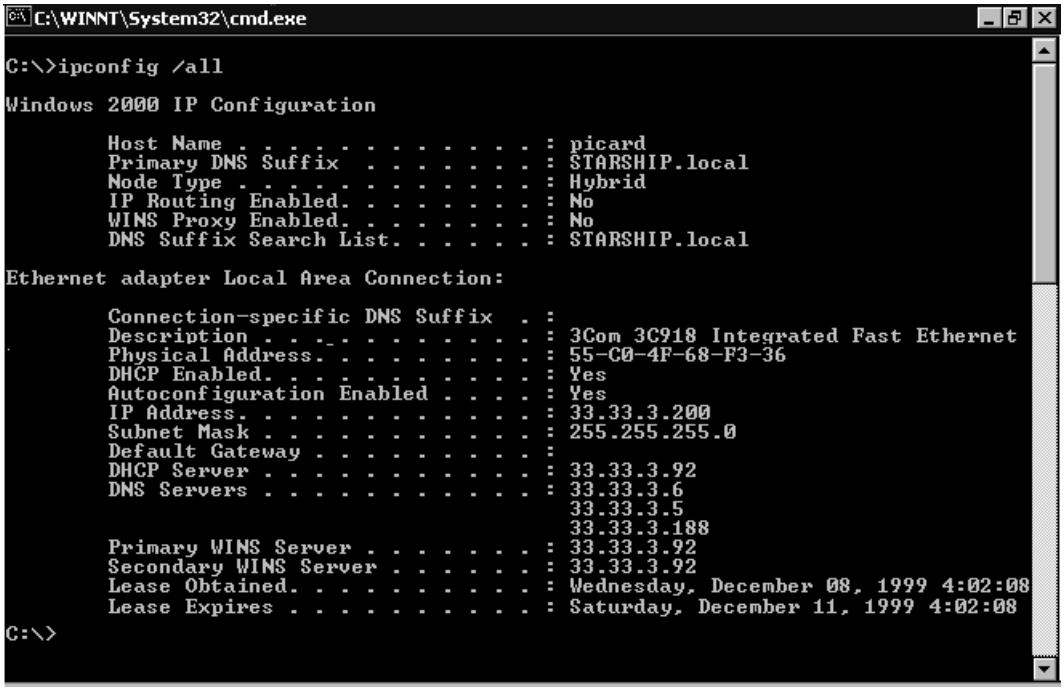
SNA Manager showing the System Identification tab



- Host Integration Server "Local" Network Name must match the AS/400 Remote network identifier.
- Host Integration Server "Local" Control Point Name must match the AS/400 Remote control point.

Here is a command window on the Host Integration Server computer. The command **ipconfig/all** has been entered to display the Host Integration Server computer's local network adapter address.

The command window



The network adapter address is also referred to as the Physical Address.

It is also possible to enter the command **net config workstation**.

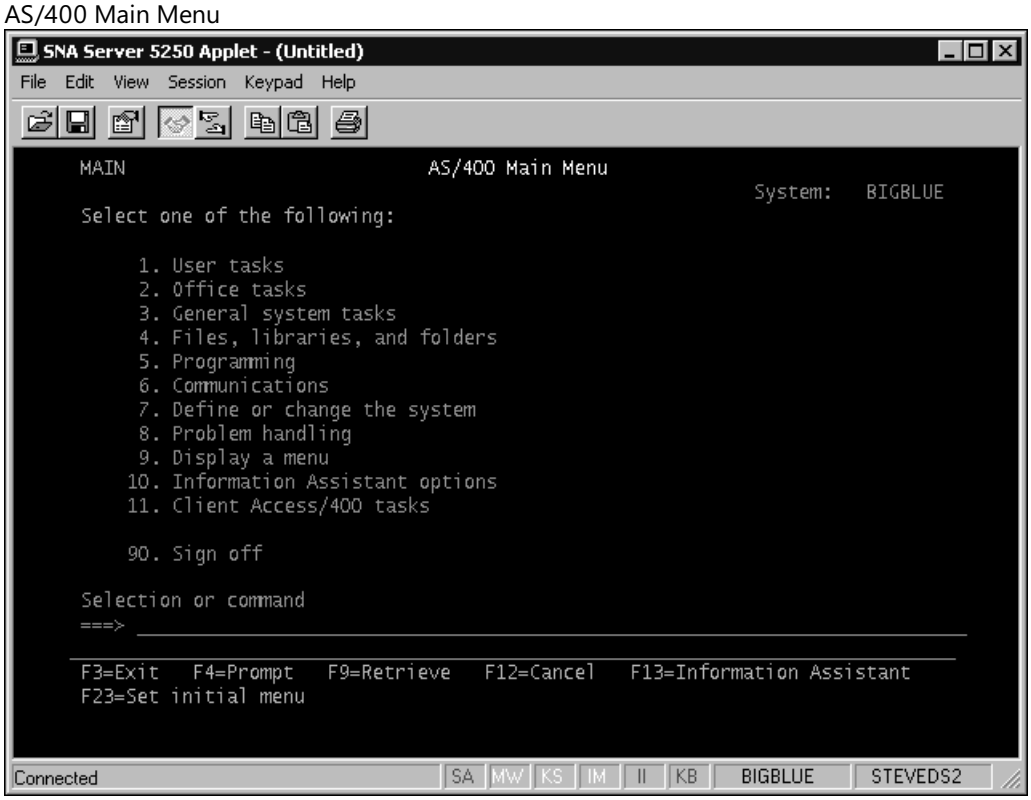
Both methods will work, but only if TCP/IP is bound to the adapter that Host Integration Server (and DLC) is using, for example, a NetBIOS-compatible transport is bound to the address used by the Host Integration Server computer. If it is not bound to it, then there is no easy way to determine the address.

Another way of saying this is: Assume there are two network adapter cards in the Host Integration Server computer. One is used for Windows domain connectivity, the other, to communicate with an AS/400. Since **ipconfig /all** will only report the address of the card used for domain connectivity, it will be difficult to find the address used for AS/400 connectivity.

AS/400 Screen Walkthrough

The starting point on the AS/400 is the **AS/400 Main Menu**:

AS/400 Main Menu



From here, you can display/work with Line and Controller descriptions, and modify configuration status fields.

What would you like to do?

Lines:

1. Display Line Descriptions
2. Change Line Descriptions
3. Change Line Status

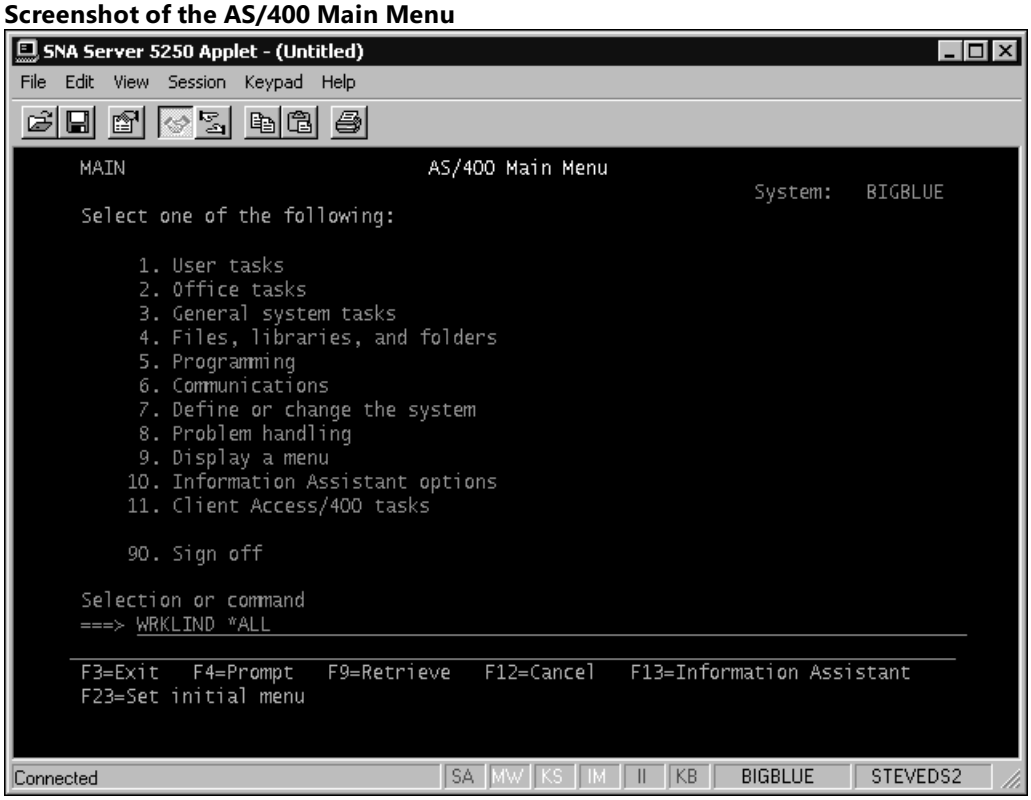
Controllers:

1. Display Controller Descriptions
2. Change Controller Descriptions

Display Line Descriptions

Open the **AS/400 Main Menu**:

Screenshot of the AS/400 Main Menu

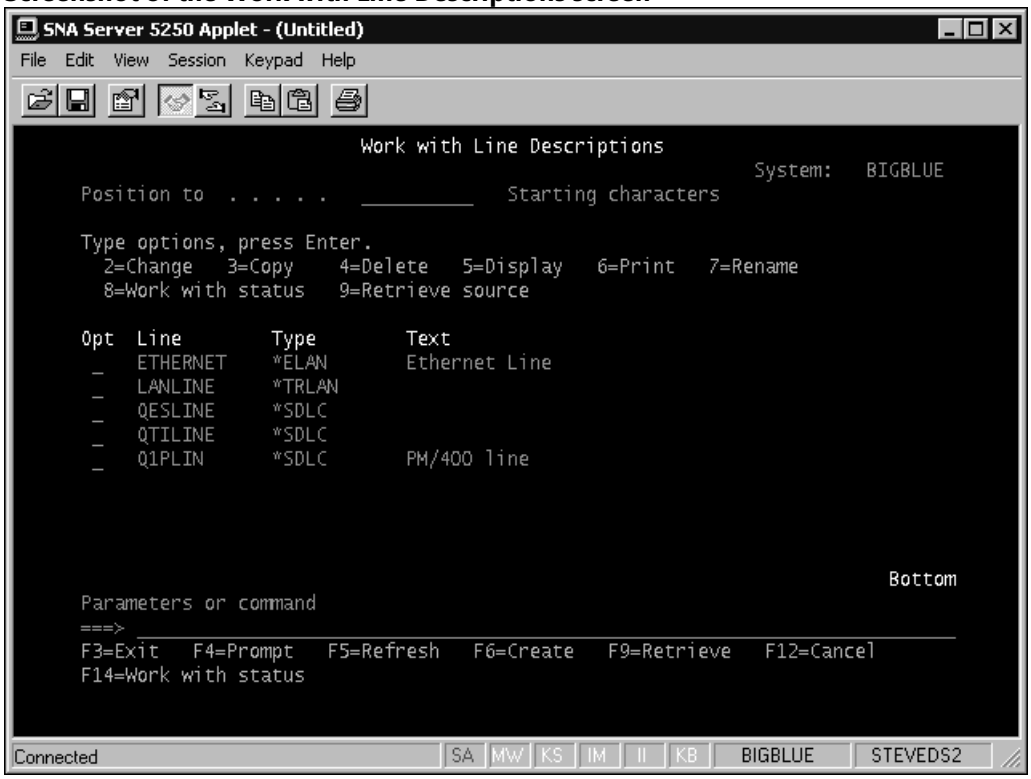


Enter **WRKLIND *ALL** at the command prompt.

Note
The AS/400 command line is case-insensitive.

This will bring up the **Work With Line Descriptions** screen:

Screenshot of the Work with Line Descriptions screen



There are three types of Lines that may be displayed on this screen: Ethernet (ELAN), Token-Ring (TRLAN), and Synchronous Data Link Control (SDLC).

The line names shown here are for illustration purposes only and may not correspond to the names displayed on your AS/400 screens.

See Also

Tasks

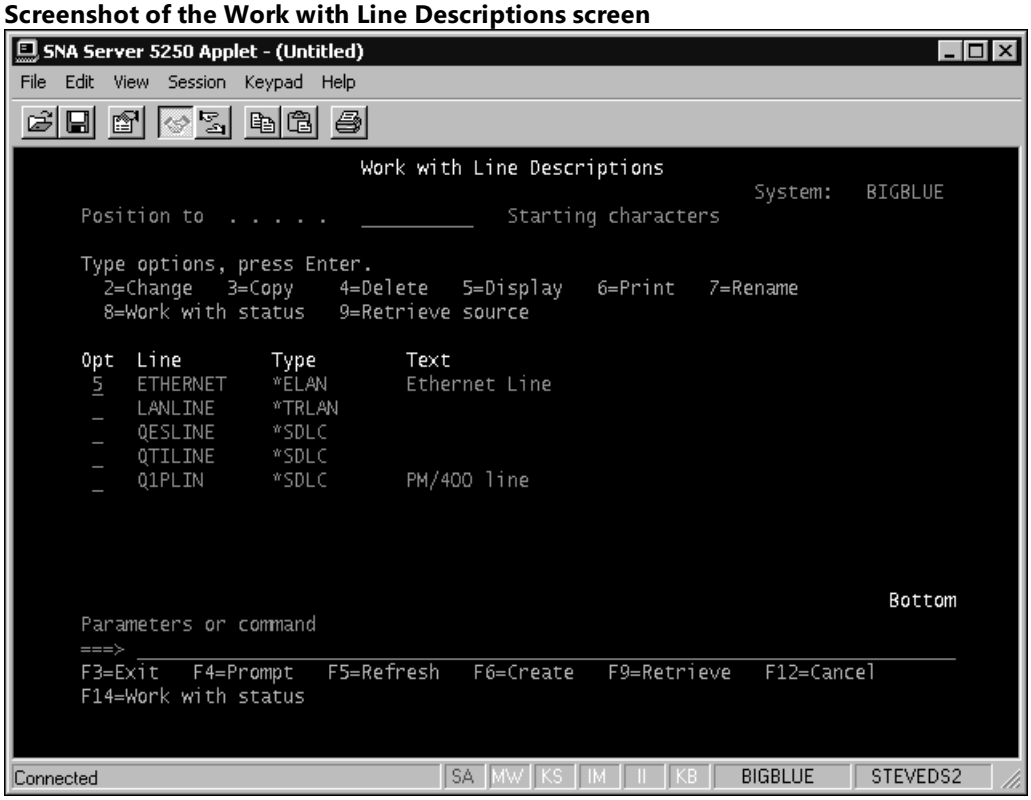
Display an Ethernet Line Description

Display a Token-Ring Line Description

Display an Ethernet Line Description

Here is the AS/400 **Work with Line Descriptions** screen:

Screenshot of the Work with Line Descriptions screen

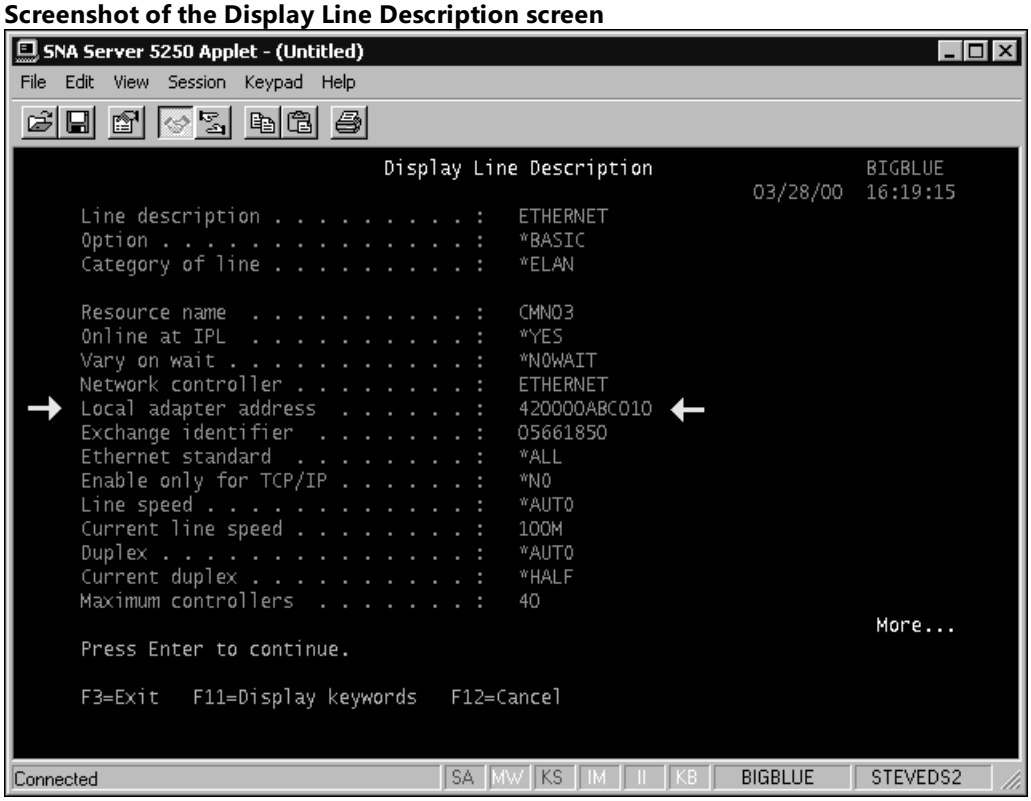


To display information for a particular Line, choose option 5 for that Line, as is shown on the screen above.

Note
The word "ETHERNET" in the "Line" column is the merely the name of a particular line, not the type of the line, which is indicated in the next column.

Here is an AS/400 **Display Line Description** screen, showing an Ethernet Line Description:

Screenshot of the Display Line Description screen



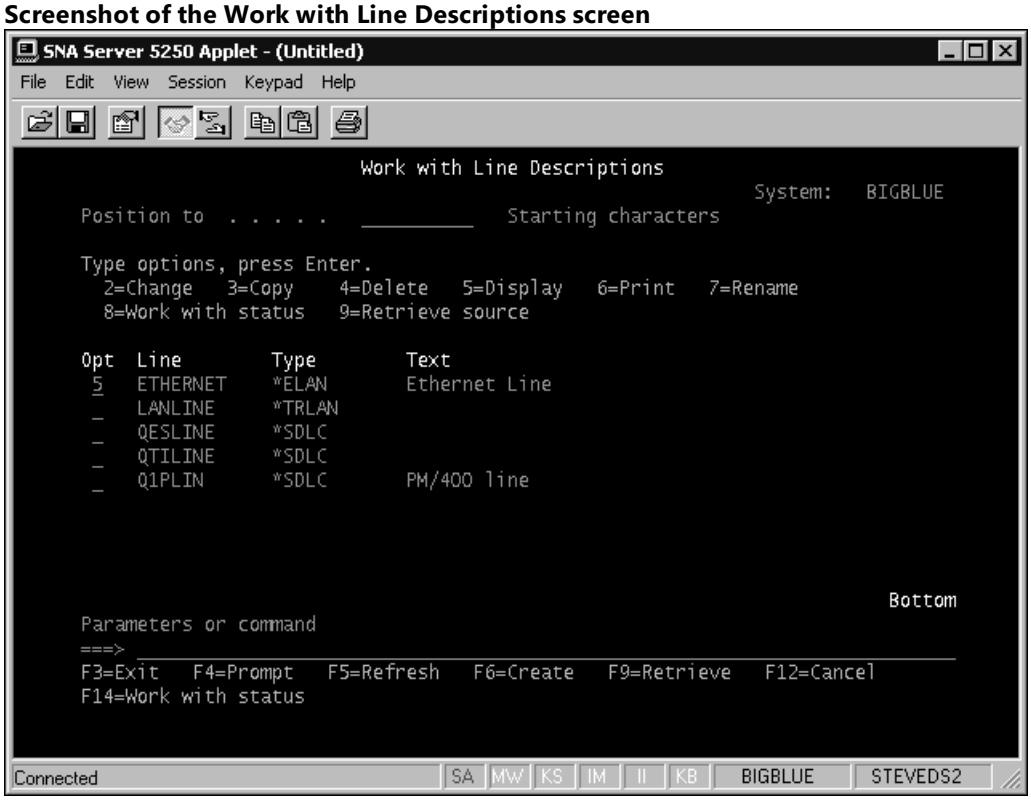
The AS/400 Ethernet network adapter address, referred to as the Local adapter address, is displayed on this screen.

This value must match the **Remote Network Address** shown on the Host Integration Server **Connection Properties** dialog box, **Address** tab.

Display a Token-Ring Line Description

Here is the **AS/400 Work with Line Descriptions** screen:

Screenshot of the Work with Line Descriptions screen

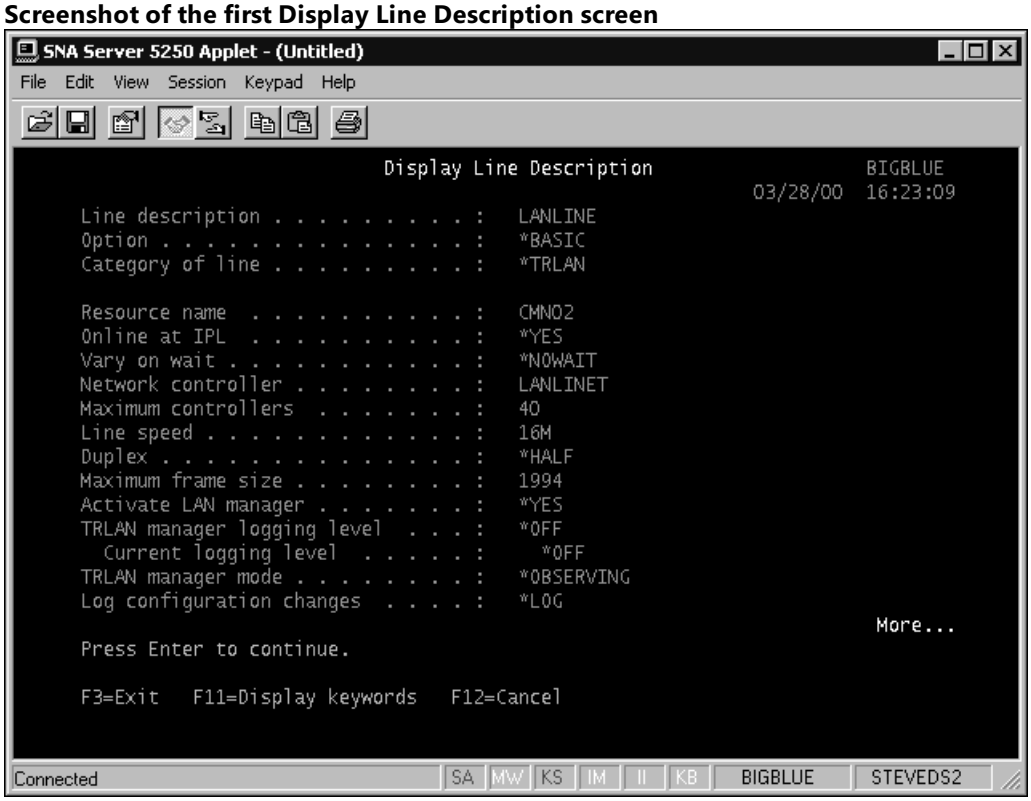


To display information for a particular Line, choose option 5 for that Line, as is shown on the screen above.

Note
The word "LANLINE" in the "Line" column is the merely the name of a particular line, not the type of the line, which is indicated in the next column.

This is the first of two **AS/400** Token-Ring **Display Line Description** screens:

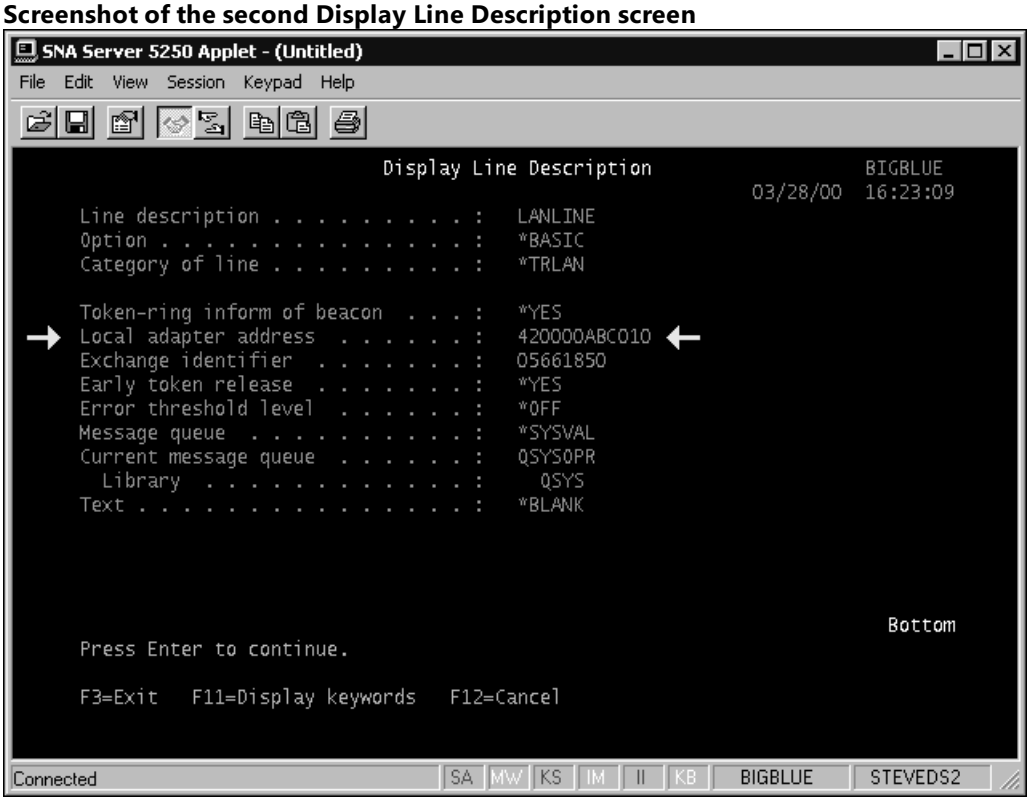
Screenshot of the first Display Line Description screen



Press the **PAGEDOWN** key to scroll to the second screen.

Here is the second Token-Ring **Display Line Description** screen:

Screenshot of the second Display Line Description screen



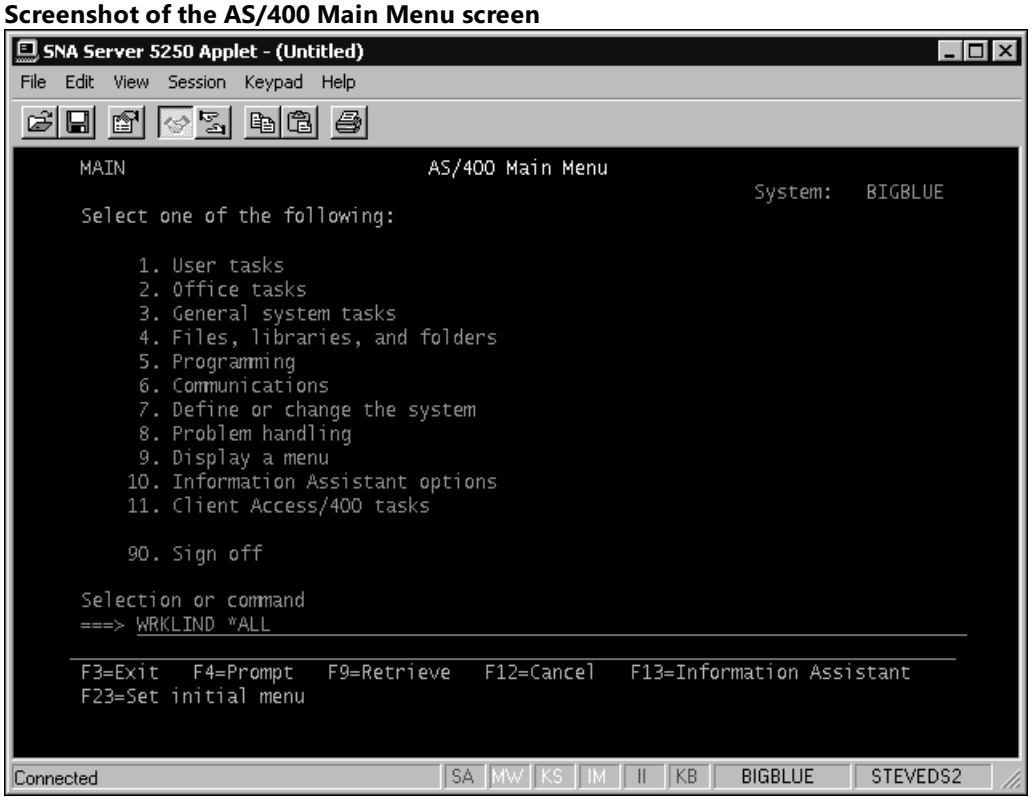
The AS/400 Token-Ring network adapter address, referred to as the **Local adapter address**, is displayed on this screen.

This value must match the **Remote Network Address** shown on the Host Integration Server **Connection Properties** dialog box, **Address** tab.

Change Line Descriptions

Open the **AS/400 Main Menu**:

Screenshot of the AS/400 Main Menu screen

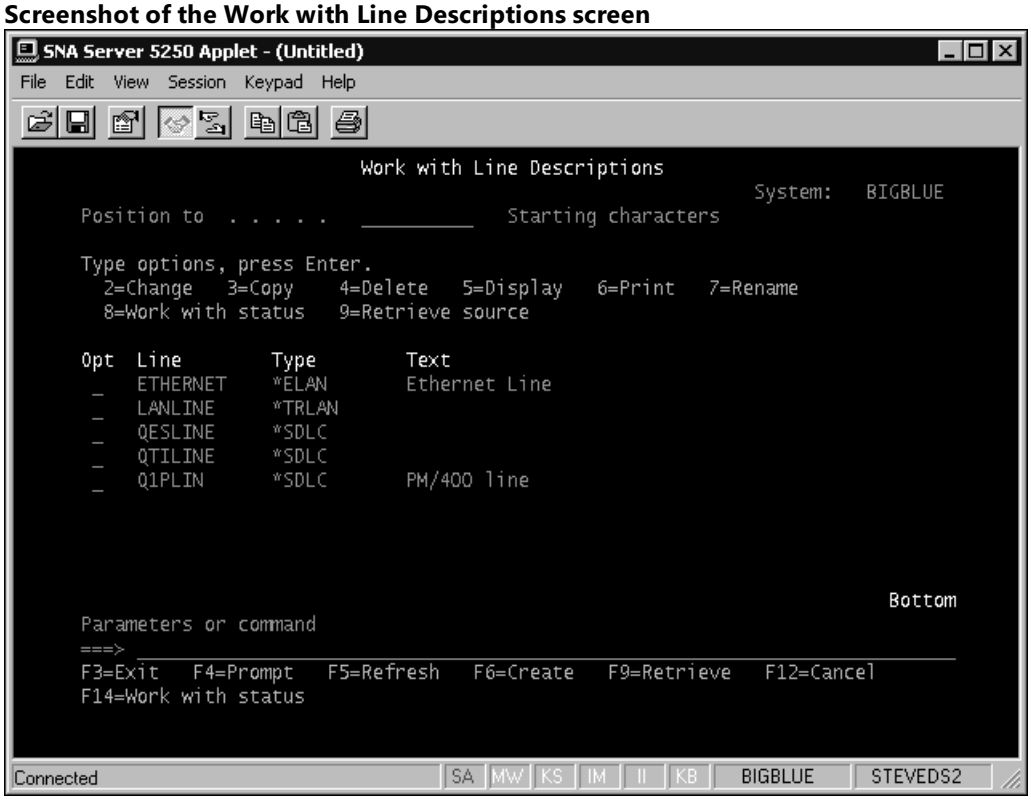


Enter **WRKLIND *ALL** at the command prompt.

Note
The AS/400 command line is not case-sensitive.

This will bring up the **Work With Line Descriptions** screen:

Screenshot of the Work with Line Descriptions screen



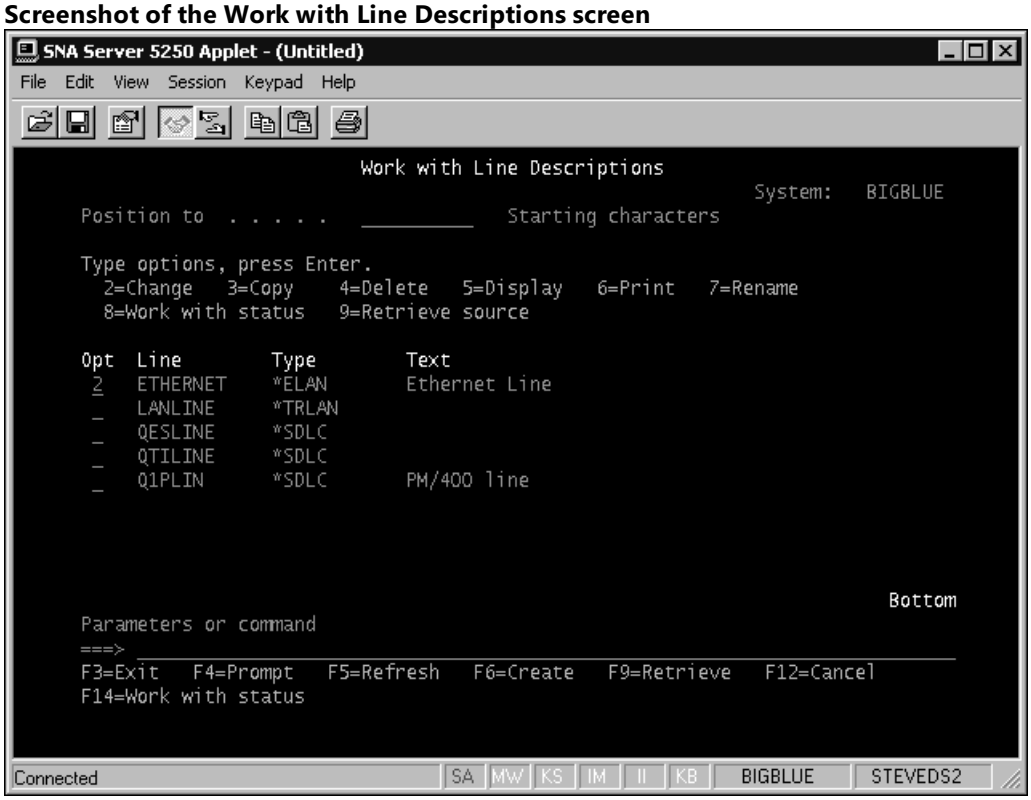
What would you like to do?

1. Change an Ethernet Line Description
2. Change a Token-Ring Line Description

Change an Ethernet Line Description

Here is the AS/400 **Work with Line Descriptions** screen:

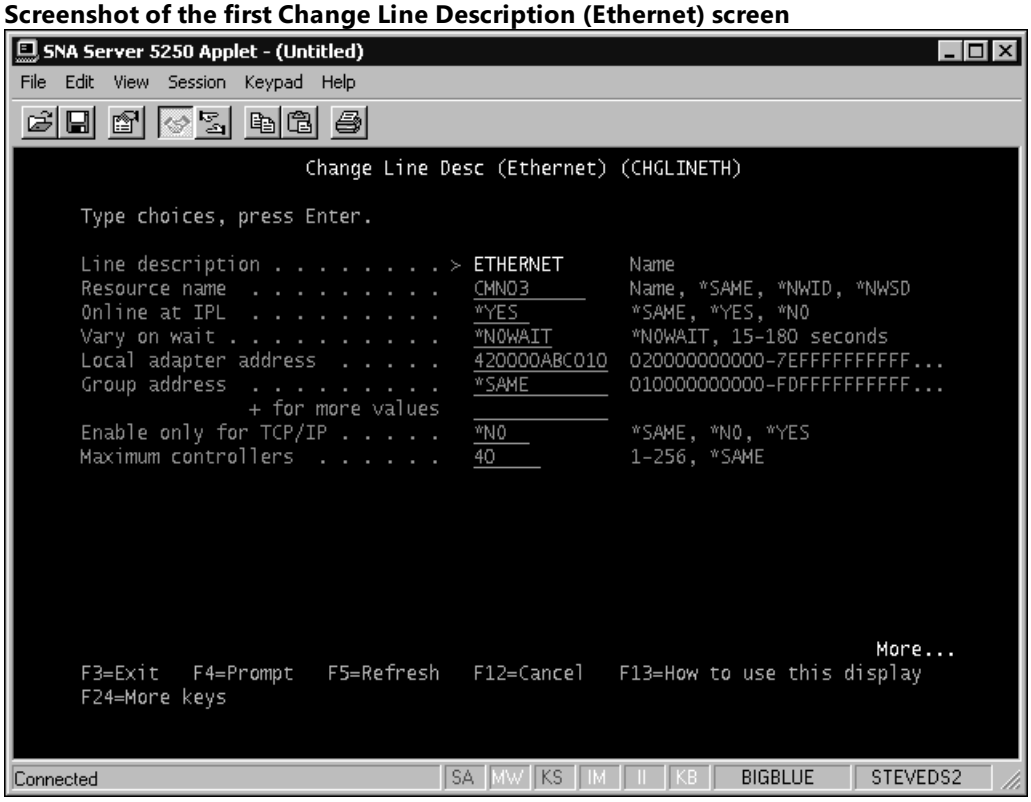
Screenshot of the Work with Line Descriptions screen



Select Option 2 (Change) to modify an Ethernet Line.

This will bring up the first of four AS/400 **Change Line Description (Ethernet)** screens:

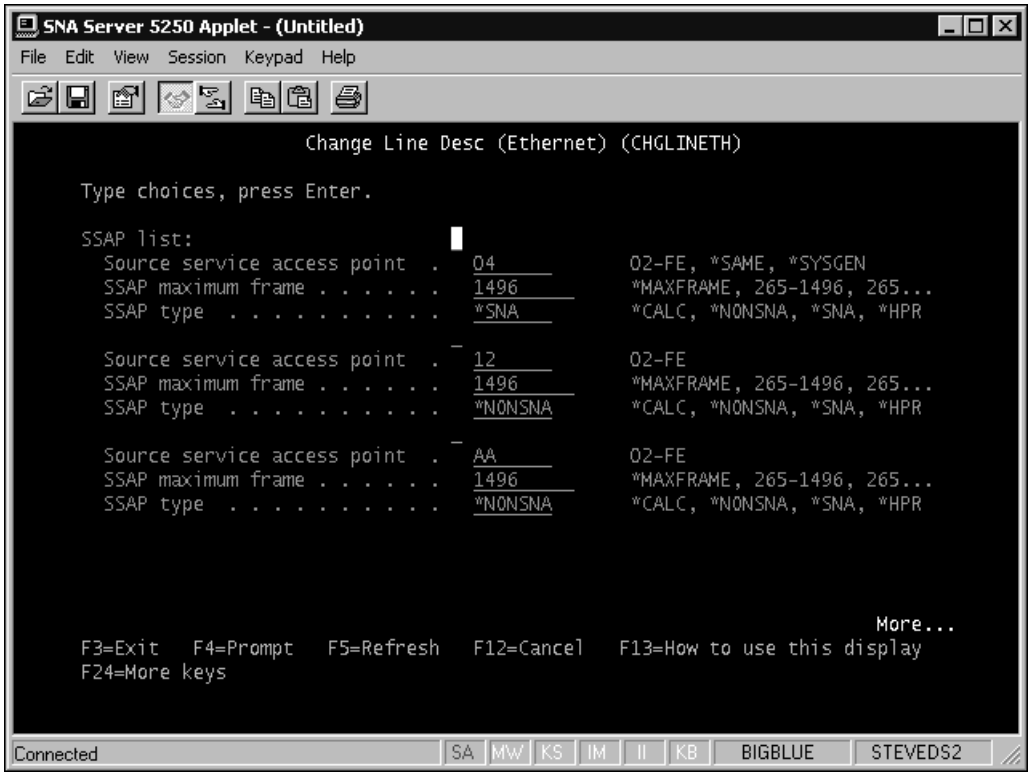
Screenshot of the first Change Line Description (Ethernet) screen



Press the **PAGEDOWN** key to scroll to the second screen.

Here is the second of four AS/400 **Change Line Description (Ethernet)** screens:

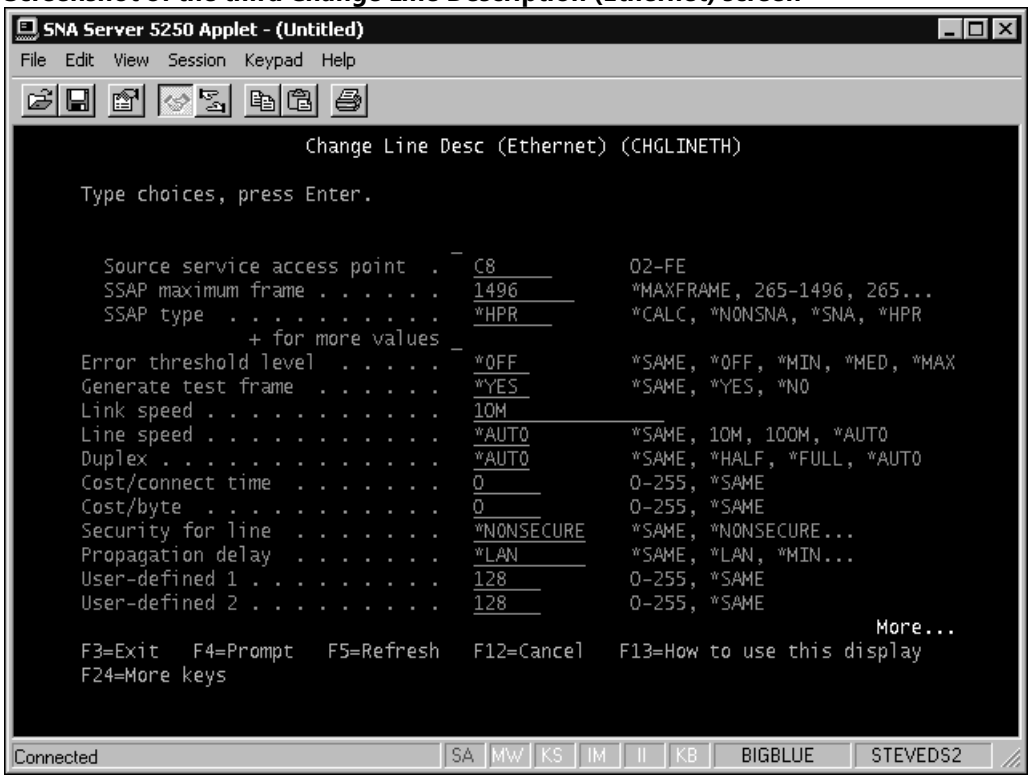
Screenshot of the second Change Line Description (Ethernet) screen



Press the **PAGEDOWN** key to scroll to the third screen.

Here is the third of four AS/400 **Change Line Description (Ethernet)** screens:

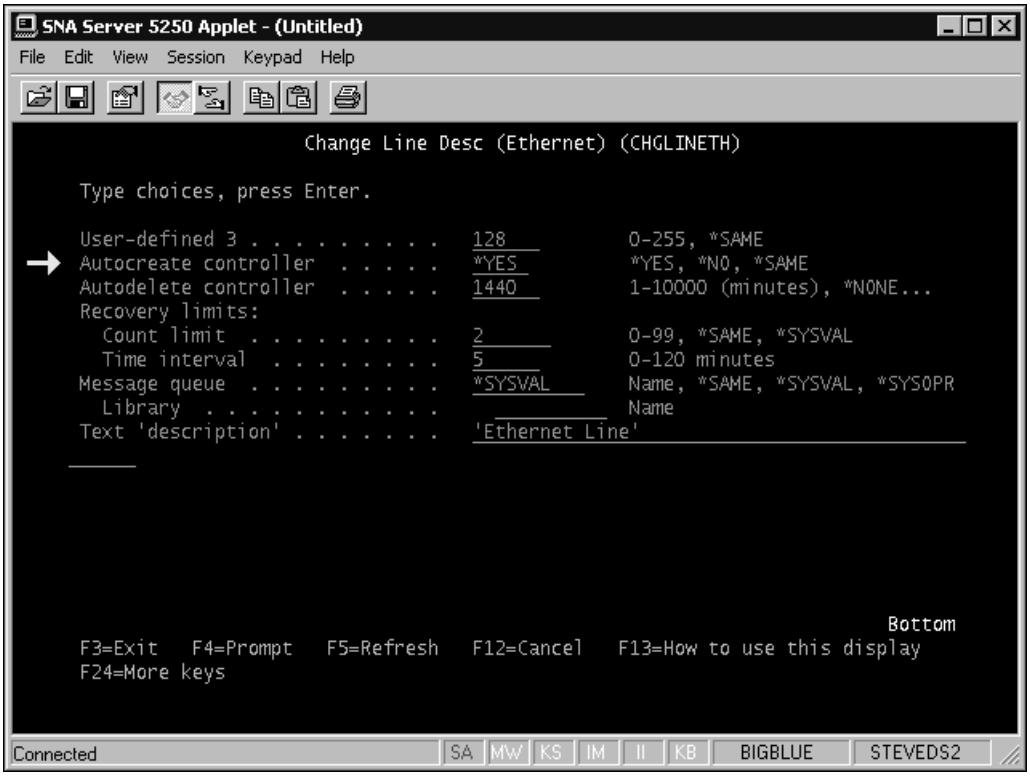
Screenshot of the third Change Line Description (Ethernet) screen



Press the **PAGEDOWN** key to scroll to the fourth screen.

Here is the last of four AS/400 **Change Line Description (Ethernet)** screens:

Screenshot of the fourth Change Line Description (Ethernet) screen

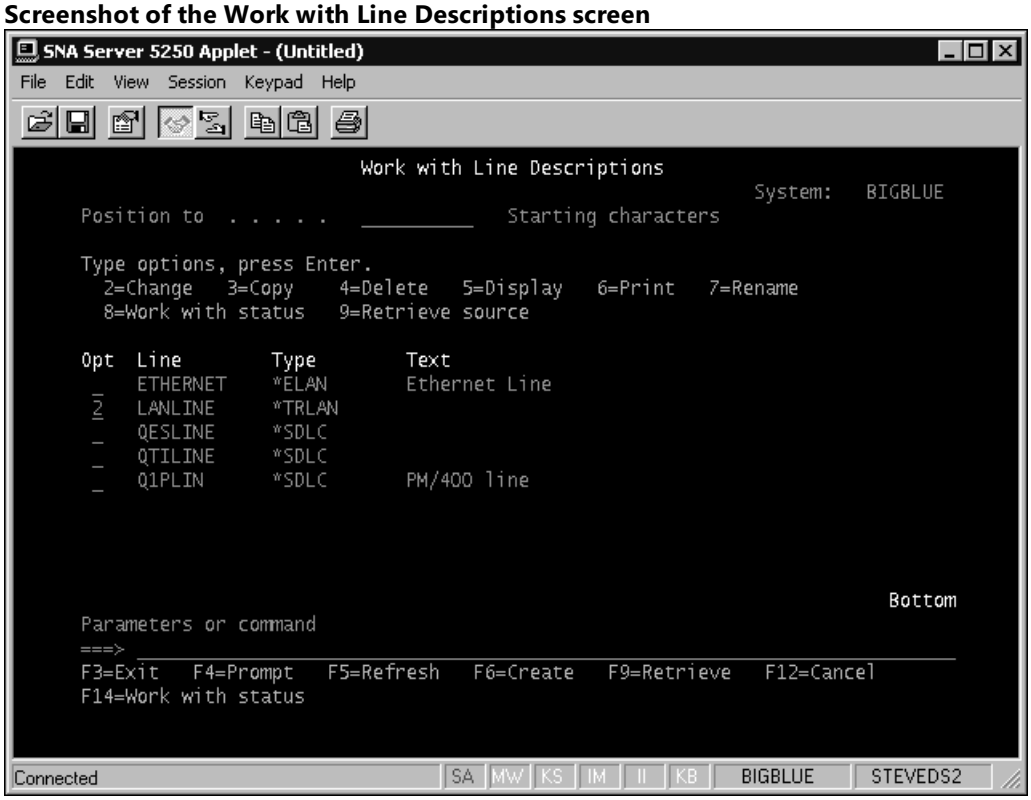


The **Autocreate controller** parameter can be set on this screen.

Change a Token-Ring Line Description

Here is the AS/400 **Work with Line Descriptions** screen:

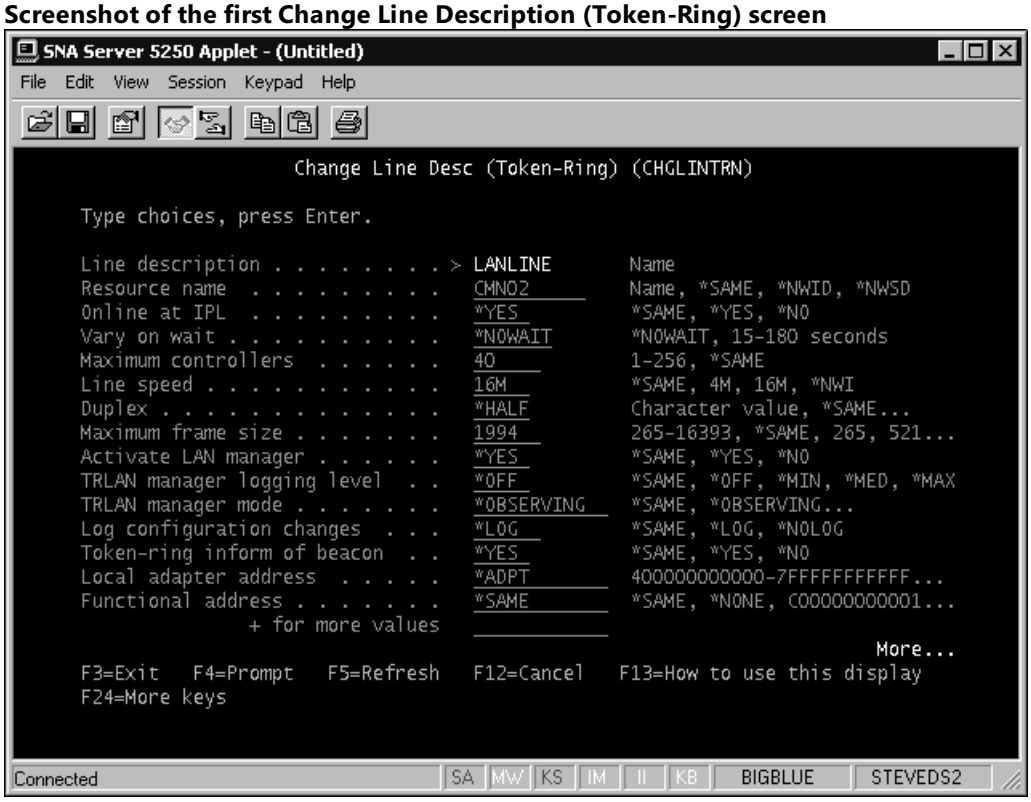
Screenshot of the Work with Line Descriptions screen



Select Option 2 (Change) to modify a Token-Ring Line.

This will bring up the first of four AS/400 **Change Line Description (Token-Ring)** screens:

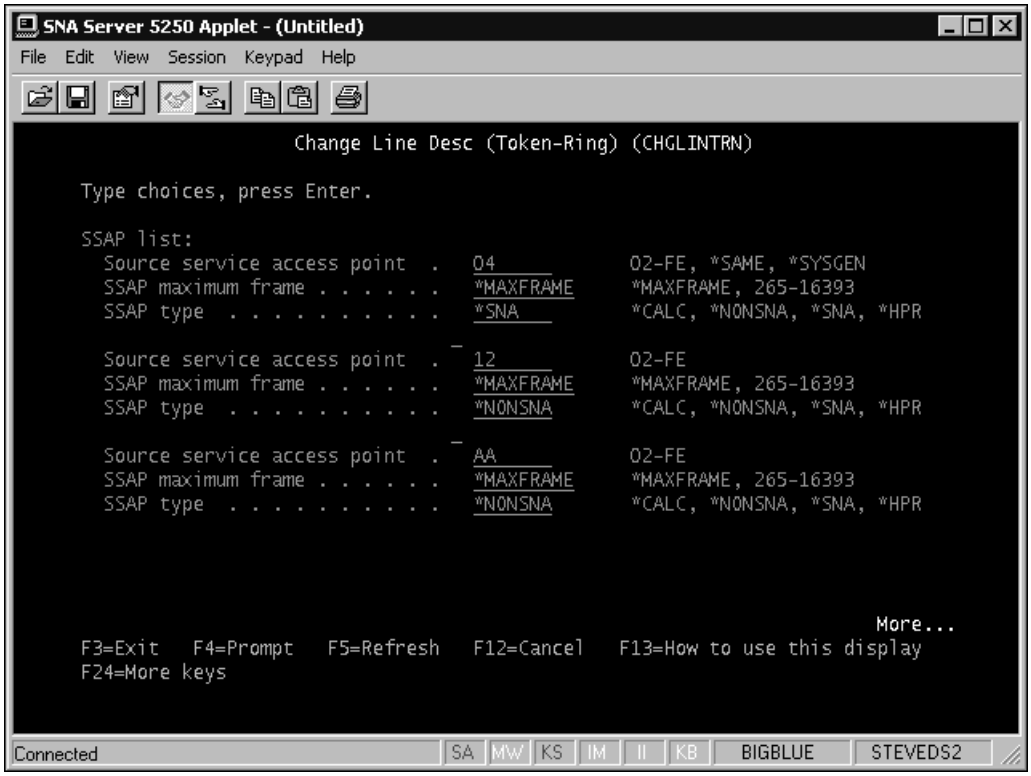
Screenshot of the first Change Line Description (Token-Ring) screen



Press the **PAGEDOWN** key to scroll to the second screen.

Here is the second of four AS/400 **Change Line Description (Token-Ring)** screens:

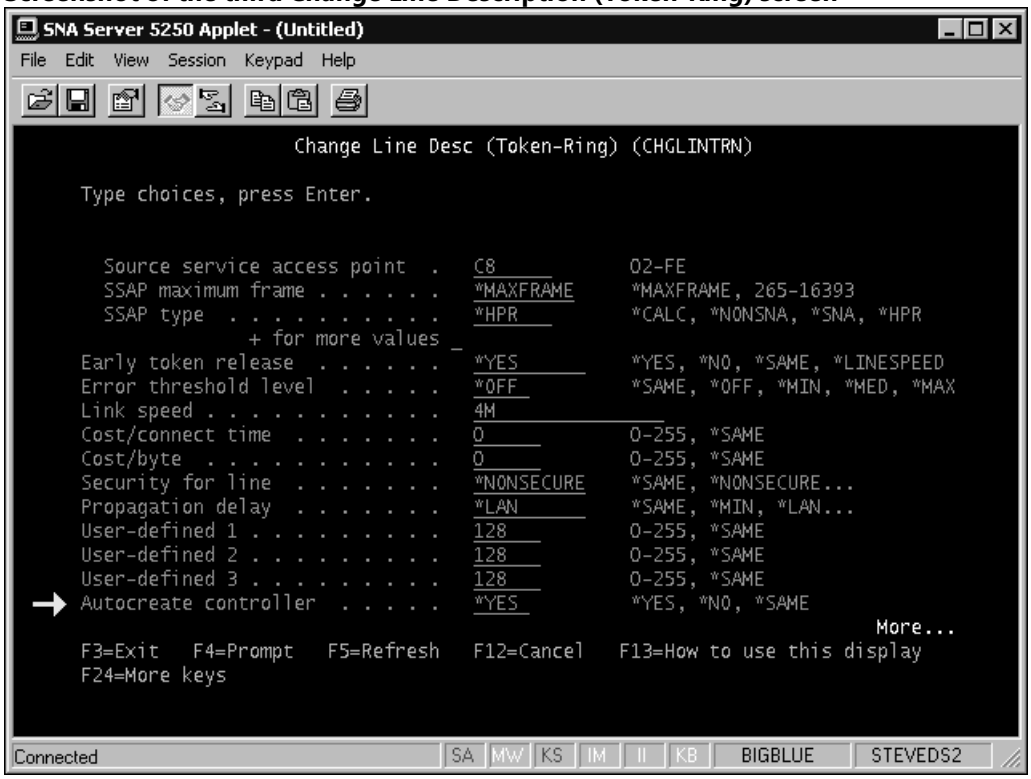
Screenshot of the second Change Line Description (Token-Ring) screen



Press the **PAGEDOWN** key to scroll to the third screen.

Here is the third of four AS/400 **Change Line Description (Token-Ring)** screens:

Screenshot of the third Change Line Description (Token-Ring) screen

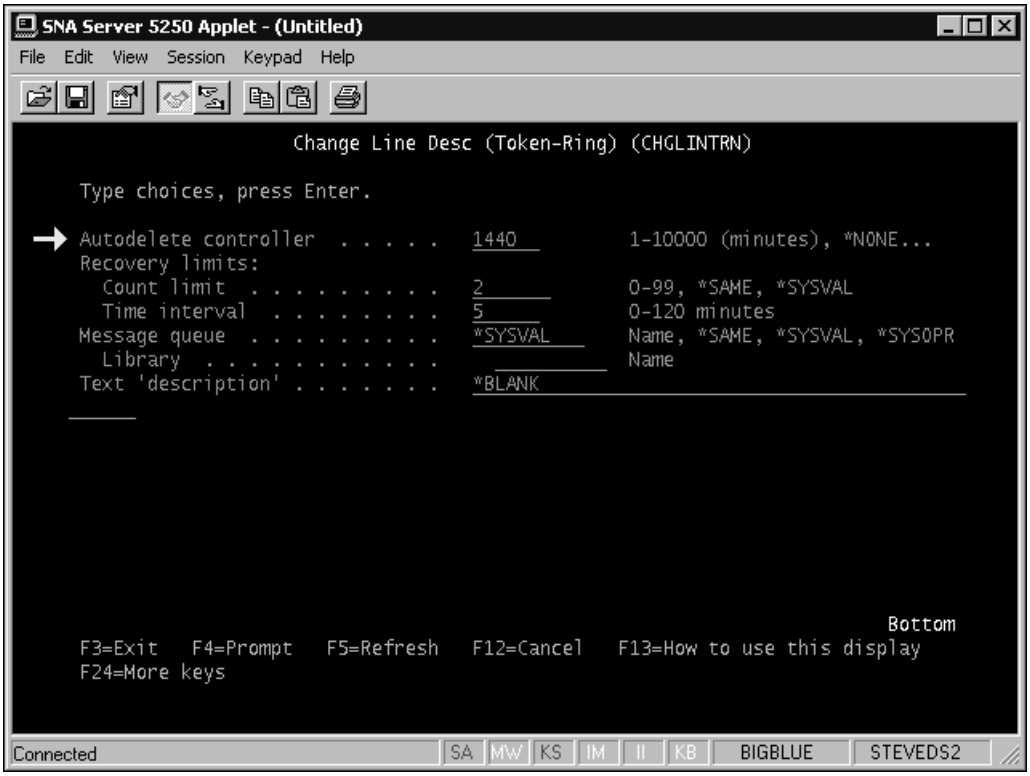


The **Autocreate controller** parameter can be set on this screen.

Press the **PAGEDOWN** key to scroll to the fourth screen.

Here is the last of four AS/400 **Change Line Description (Token-Ring)** screens:

Screenshot of the fourth Change Line Description (Token-Ring) screen

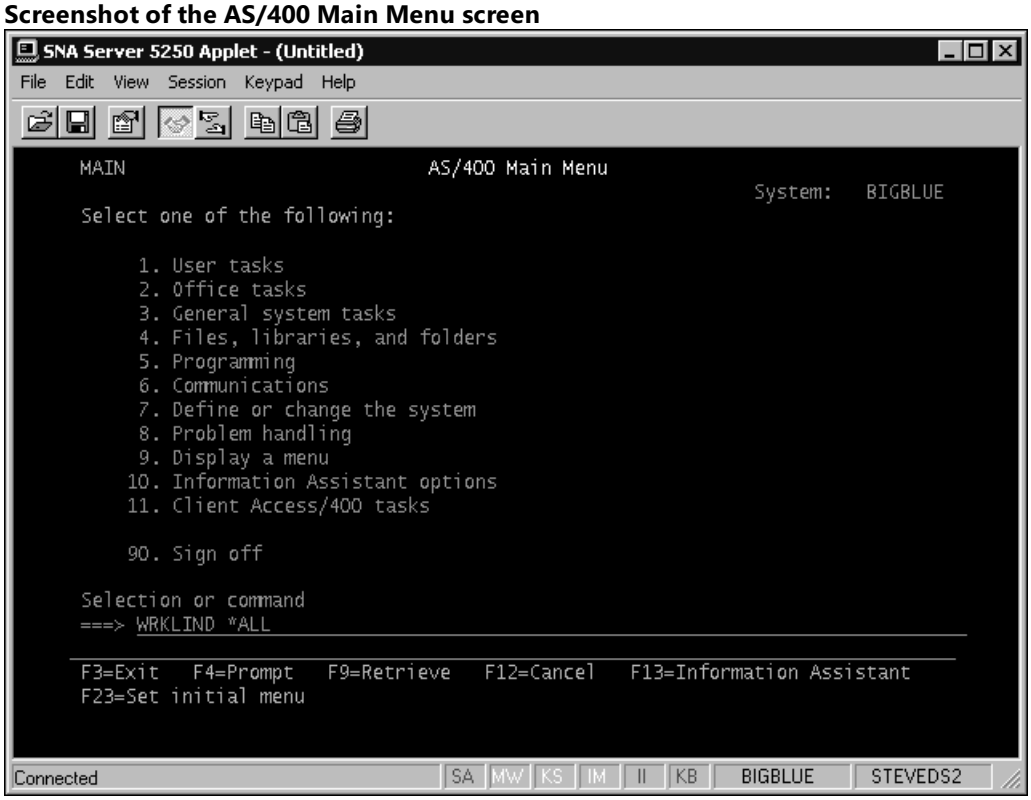


The Autodelete controller parameter can be set on this screen.

Change Line Status

Open the **AS/400 Main Menu**:

Screenshot of the AS/400 Main Menu screen

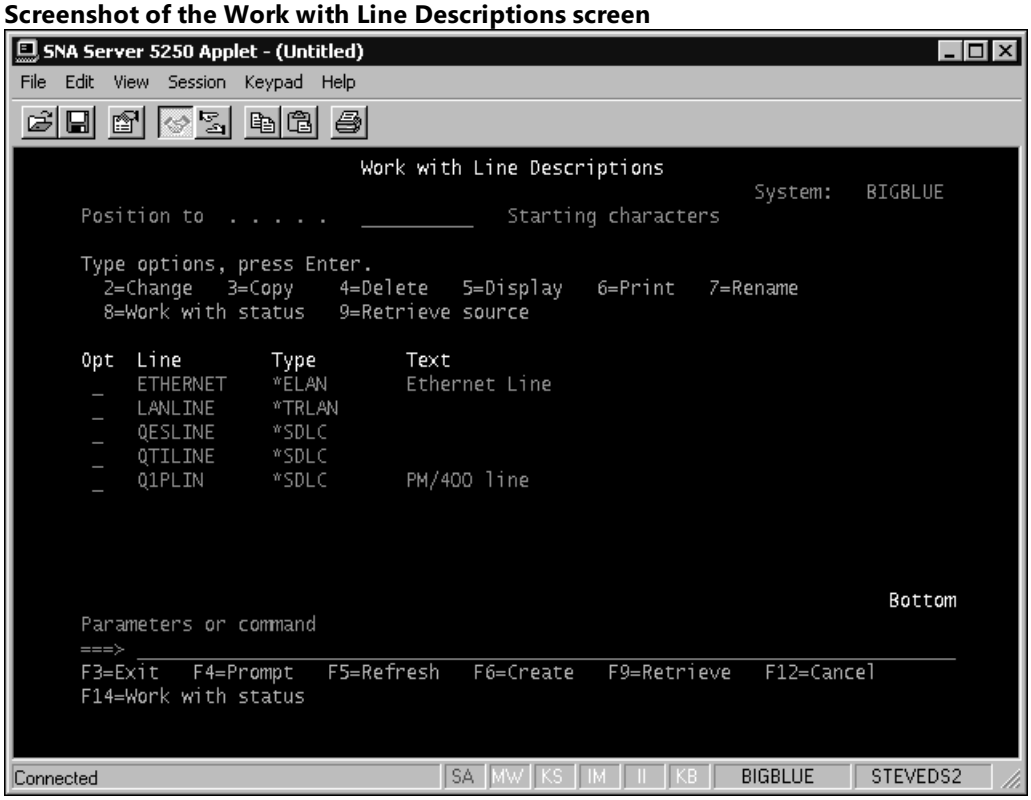


Enter **WRKLIND *ALL** at the command prompt.

Note
The AS/400 command line is not case-sensitive).

This will bring up the **Work With Line Descriptions** screen:

Screenshot of the Work with Line Descriptions screen

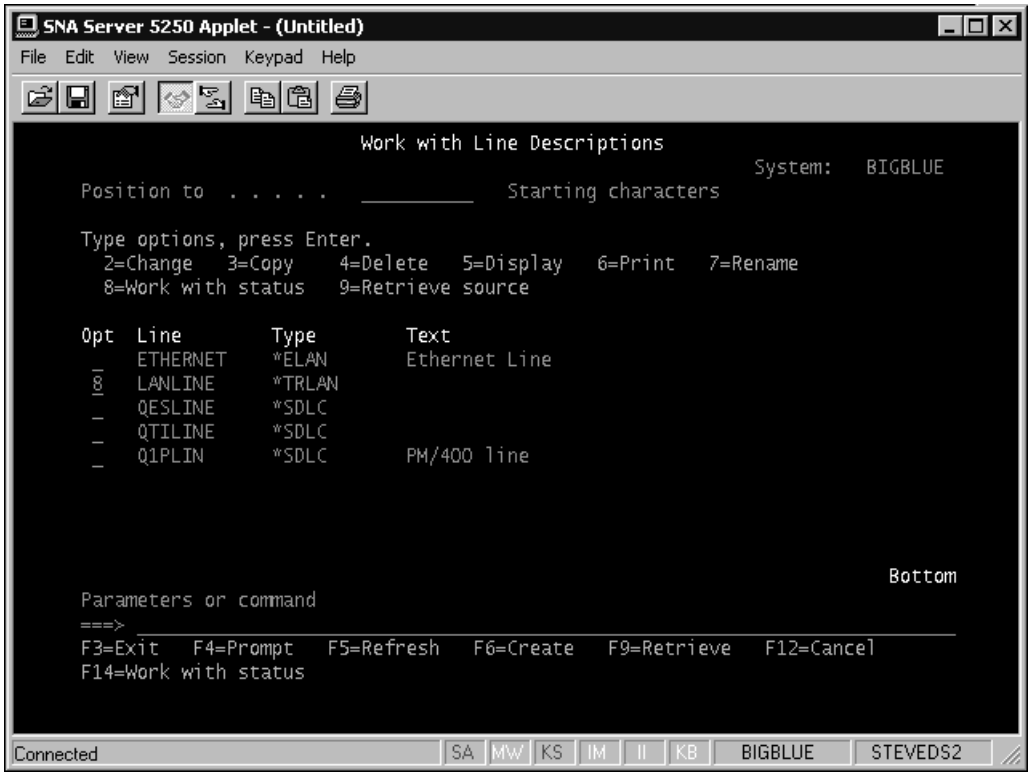


What would you like to do?

1. Change an Ethernet Line's Status
2. Change a Token-Ring Line's Status

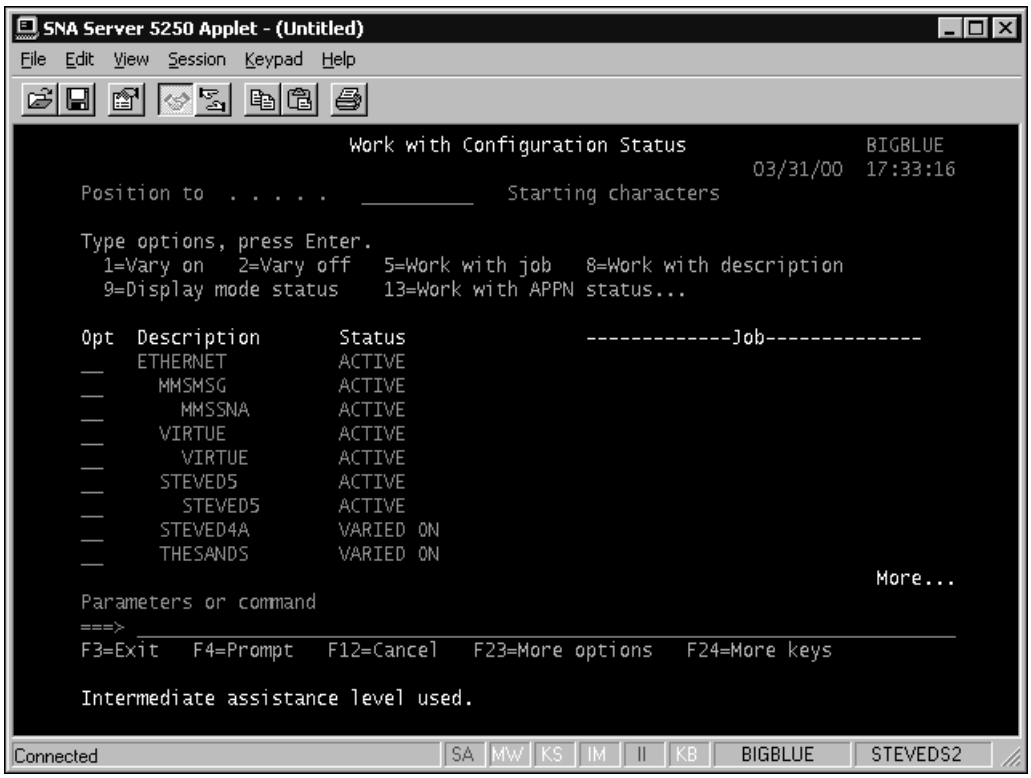
Change an Ethernet Line's Status

Here is the AS/400 **Work With Line Descriptions** screen:



Choose the **Status** option for the desired line by entering the number 8 in the **Opt** column, next to the Ethernet Line you want to change.

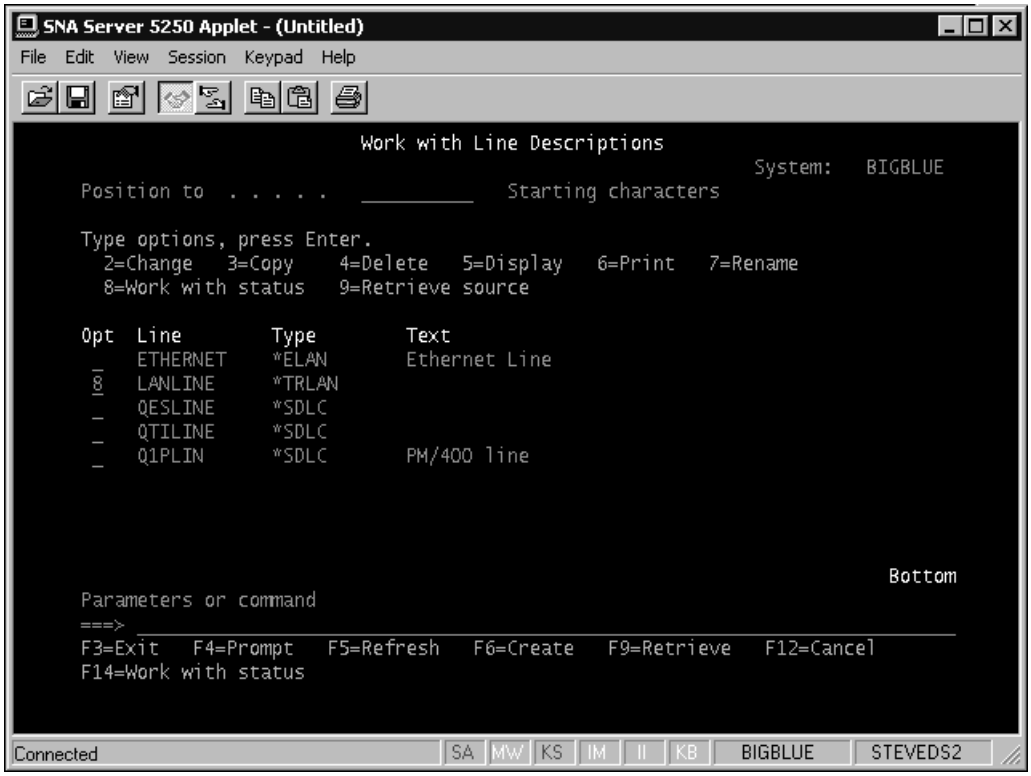
Here is the AS/400 **Work with Configuration Status** screen:



Select option 2 to vary the Line status off, and option 1 to vary the Line status back on.

Change a Token-Ring Line's Status

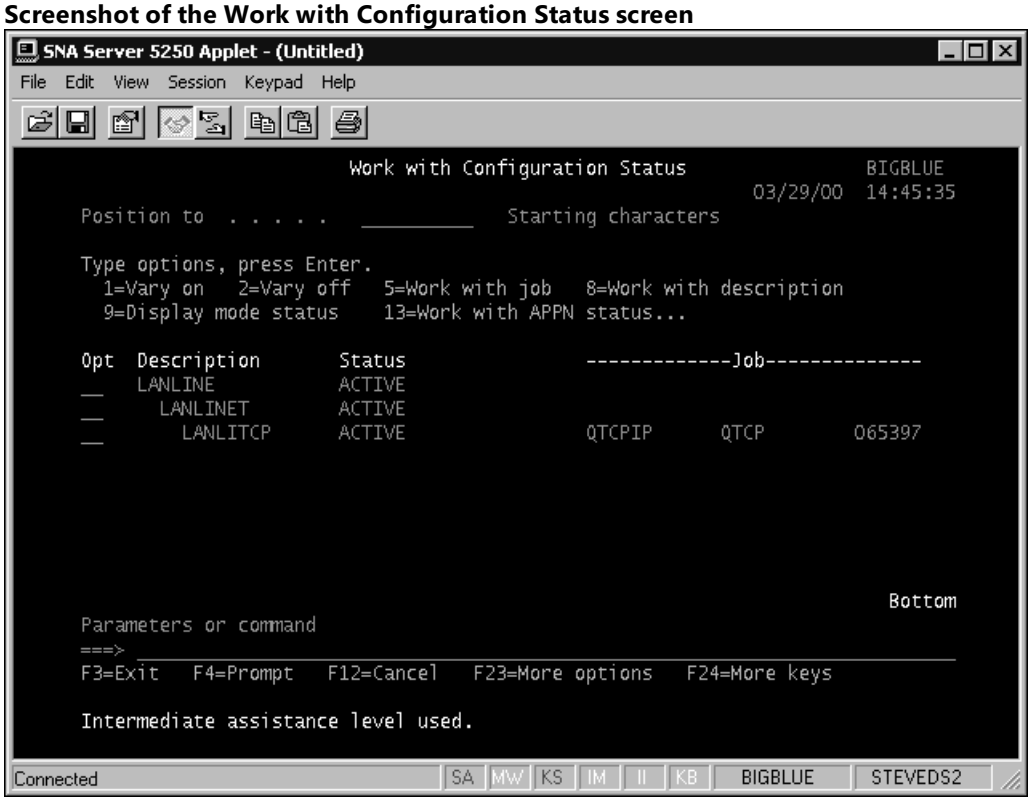
The following figure shows the AS/400 **Work With Line Descriptions** screen:



Choose the **Status** option for the desired line by entering the number **8** in the **Opt** column, next to the Token-Ring Line you want to change.

The AS/400 **Work with Configuration Status** screen appears:

Screenshot of the Work with Configuration Status screen



Select option 2 to vary the Line status off, and option 1 to vary the Line status back on.

Display Controller Descriptions

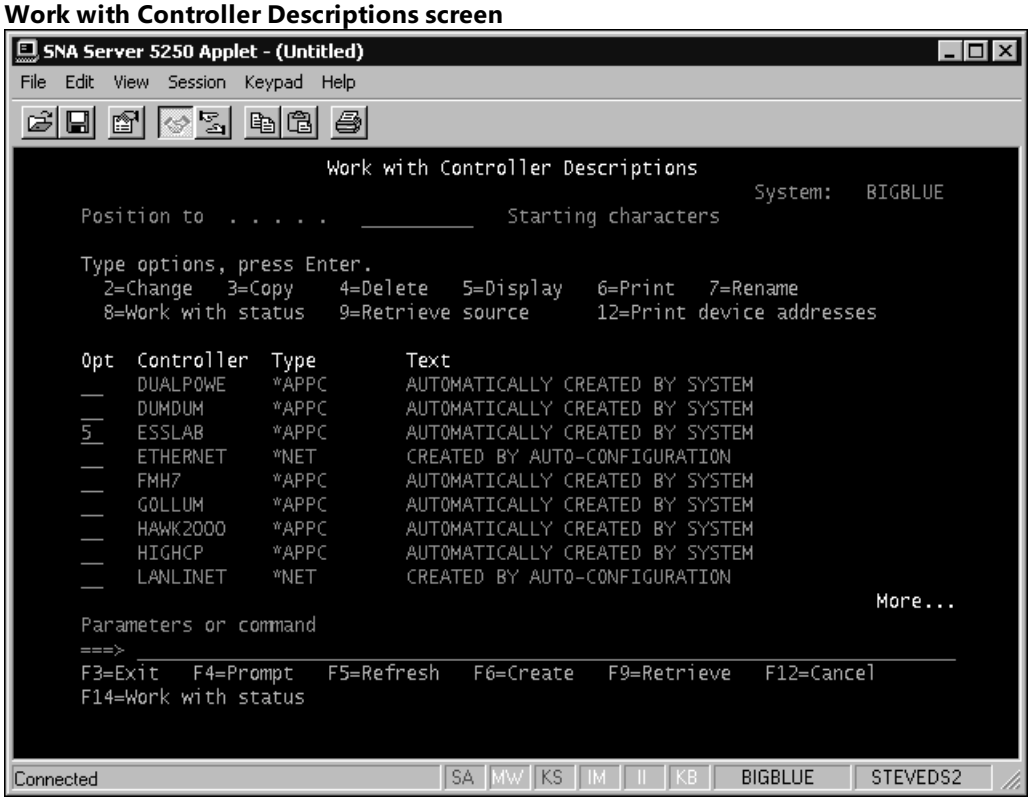
Open the **AS/400 Main Menu**:

AS/400 Main Menu screen

Enter **WRKCTLD *ALL** at the command prompt.

This will bring up the **Work With Controller Descriptions** screen, which lists available AS/400 Controller Descriptions:

Work with Controller Descriptions screen

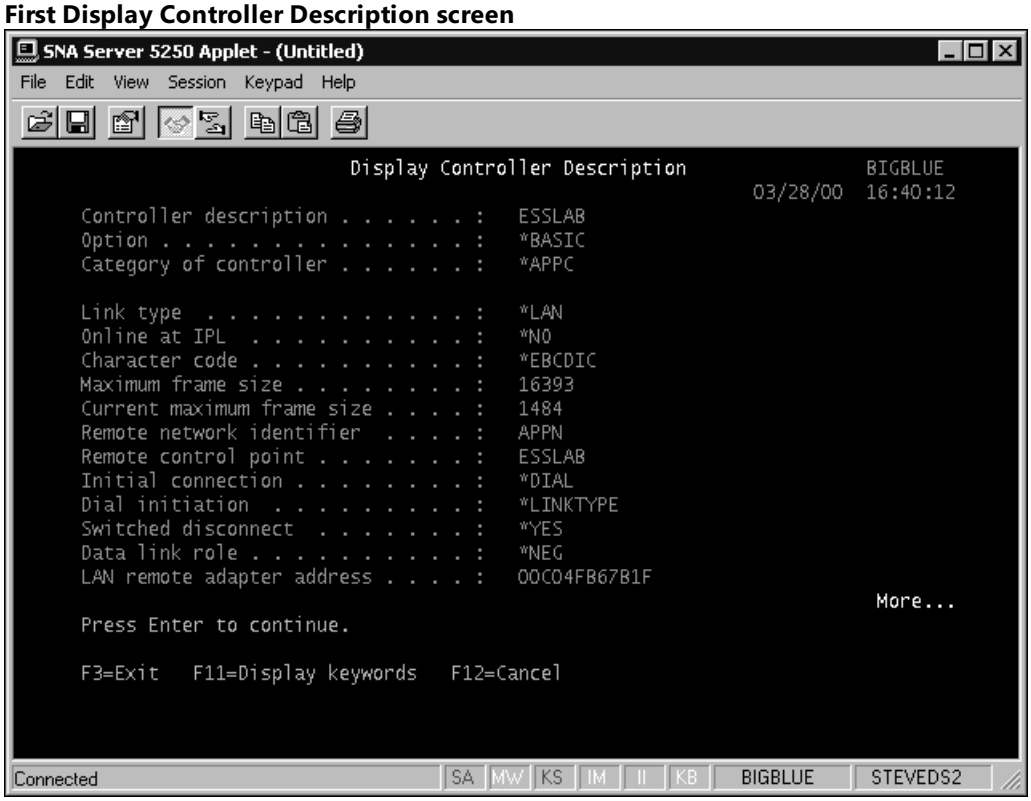


The controller names shown here are for illustration purposes only and may not correspond to the names displayed on your AS/400 screens.

Choose the **Display** option by entering the number 8 next to the Controller that you want to display.

Here is the first of two **Display Controller Description** screens:

First Display Controller Description screen



There are three important non-modifiable fields displayed on this screen:

AS/400 Field Name	Host Integration Server Field name
Remote network identifier	"Local" Network Name
Remote control point	"Local" Control Point
LAN remote adapter address	Network adapter address

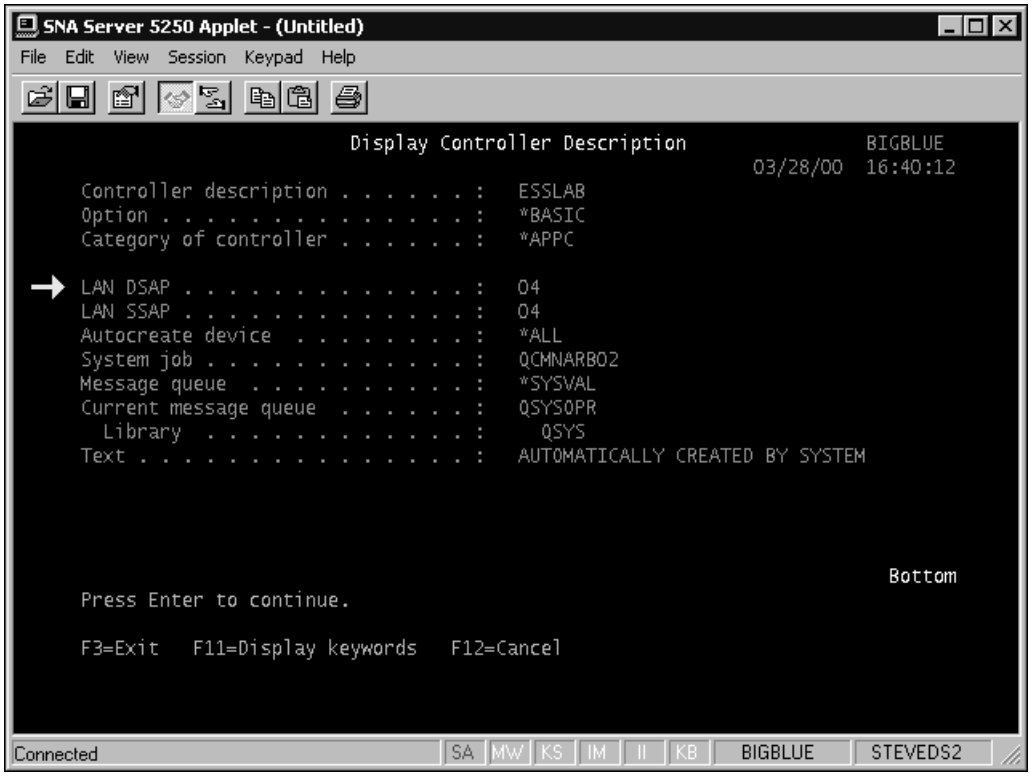
Note

The **LAN remote adapter address** corresponds to the Host Integration Server computers local network adapter address, *no t* to the AS/400s local network adapter address.

Press the **PAGEDOWN** key to display the second **Display Controller Description** screen.

Here is the second **Display Controller Description** screen:

Second Display Controller Description screen



There are two important non-modifiable fields on this screen:

AS/400 Field Name	Host Integration Server Field name
LAN DSAP	Local SAP Address
LAN SSAP	Remote SAP Address

Change Controller Descriptions

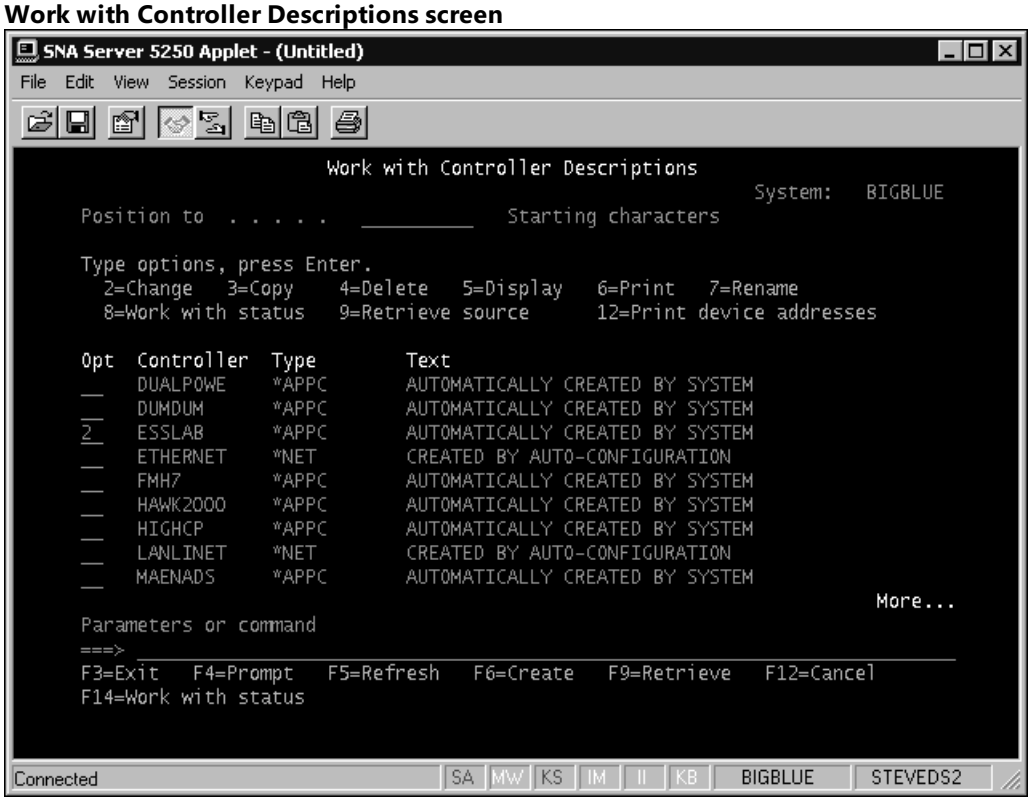
Open the **AS/400 Main Menu**:

AS/400 Main Menu screen

Enter **WRKCTLD *ALL** at the command prompt.

This will bring up the **Work With Controller Descriptions** screen, which lists available AS/400 Controller Descriptions:

Work with Controller Descriptions screen

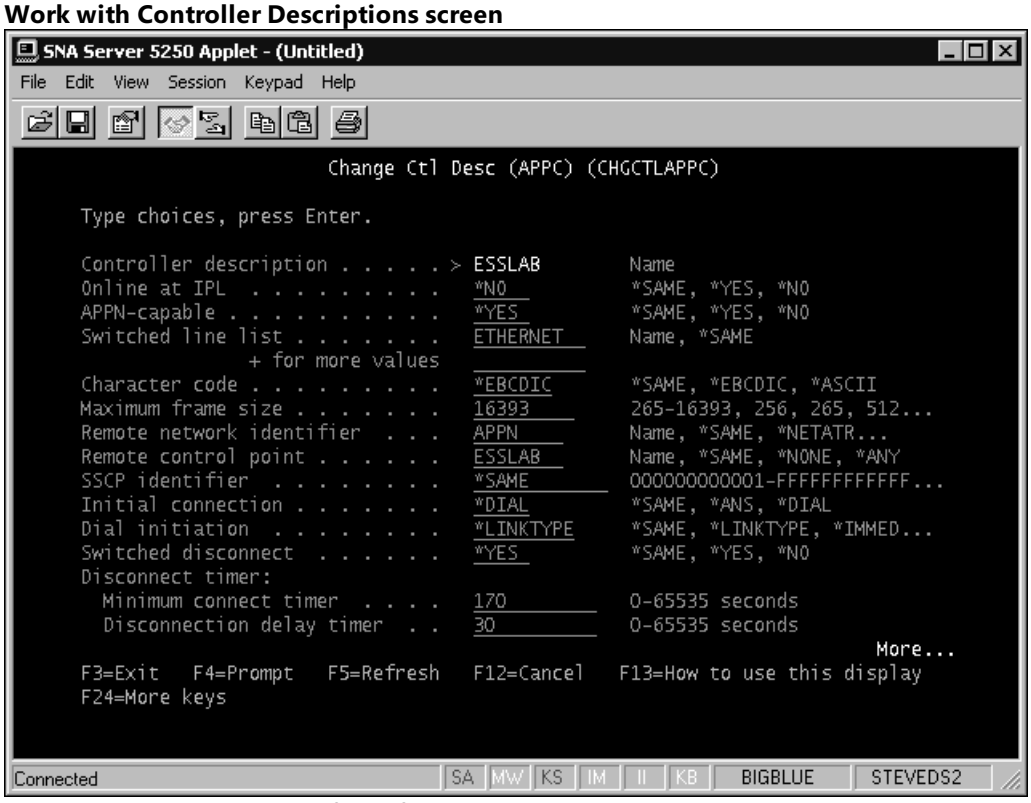


The controller names shown here are for illustration purposes only and may not correspond to the names displayed on your AS/400 screens.

Select the **Change** option (option 2) for the Controller you want to change.

Here is the first of four AS/400 **Chng Ctl Desc (APPC)** screens:

Work with Controller Descriptions screen



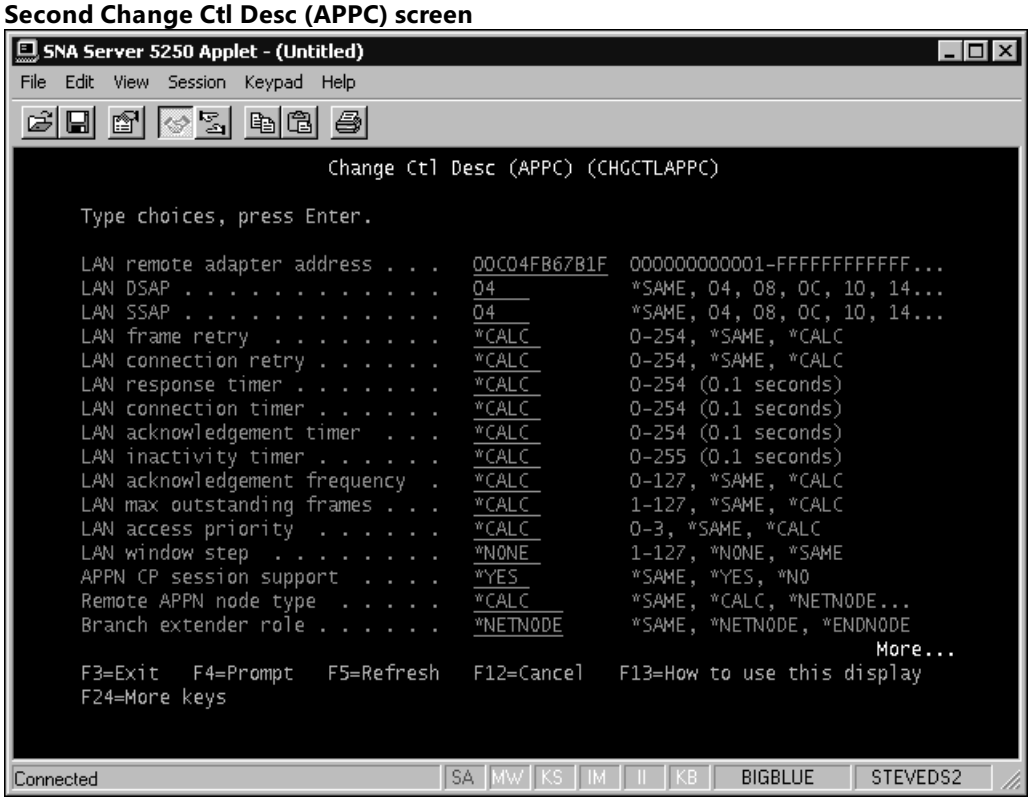
There are two important modifiable fields on this screen:

AS/400 Field Name	Host Integration Server Field name
Remote network identifier	"Local" Network Name
Remote control point	"Local" Control Point Name

Press the **PAGEDOWN** key to scroll to the second screen.

Here is the second of four **Chng Ctl Desc (APPC)** screens:

Second Change Ctl Desc (APPC) screen



There are three important modifiable fields on this screen:

AS/400 Field Name	Host Integration Server field name

LAN remote adapter address	Network adapter address (see Note 2)
LAN DSAP	**Local SAP Address**
LAN SSAP	**Remote SAP Address**

Note

The Host Integration Server computers **Remote Network Address MUST NOT** match the AS/400 controllers **LAN remote adapter address**. The AS/400 controllers **LAN remote adapter address** is the Host Integration Server computers local net work (MAC) address. This is a common mistake.

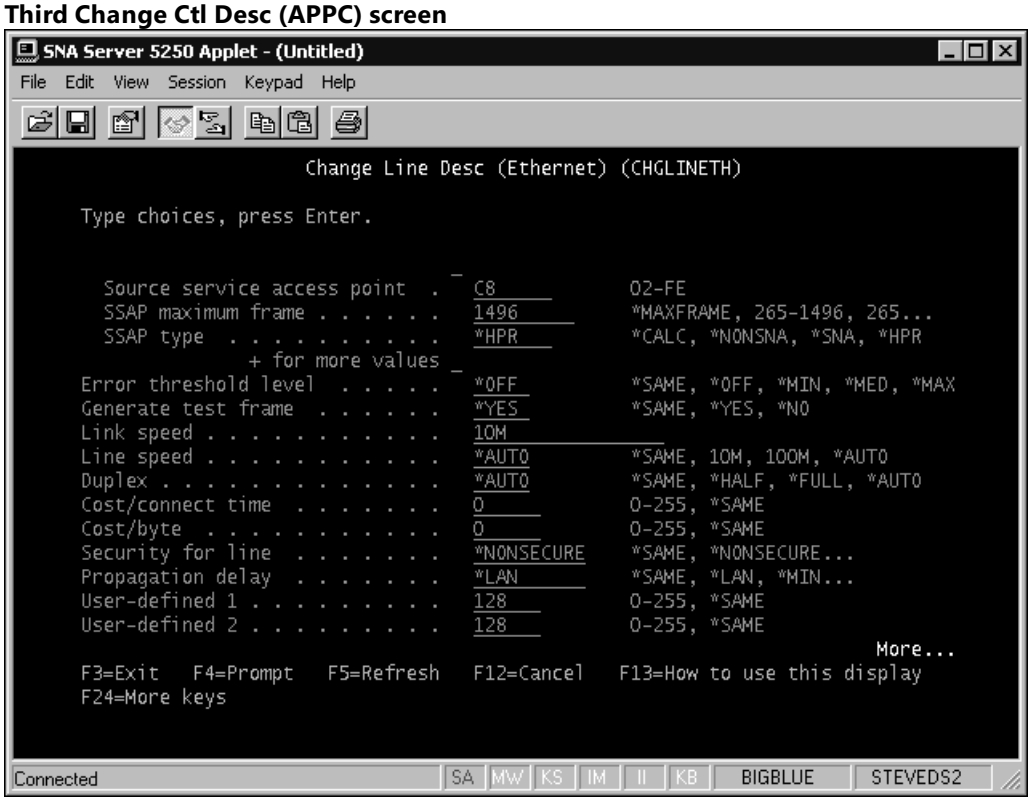
Note

The field name "Network adapter address" does not actually exist within Host Integration Server. This is why it is not in bold t ype. This field name refers to the address of the network adapter on the Host Integration Server computer.

Press the **PAGEDOWN** key to scroll to the third screen.

Here is the third of four AS/400 **Chng Ctl Desc (APPC)** screens:

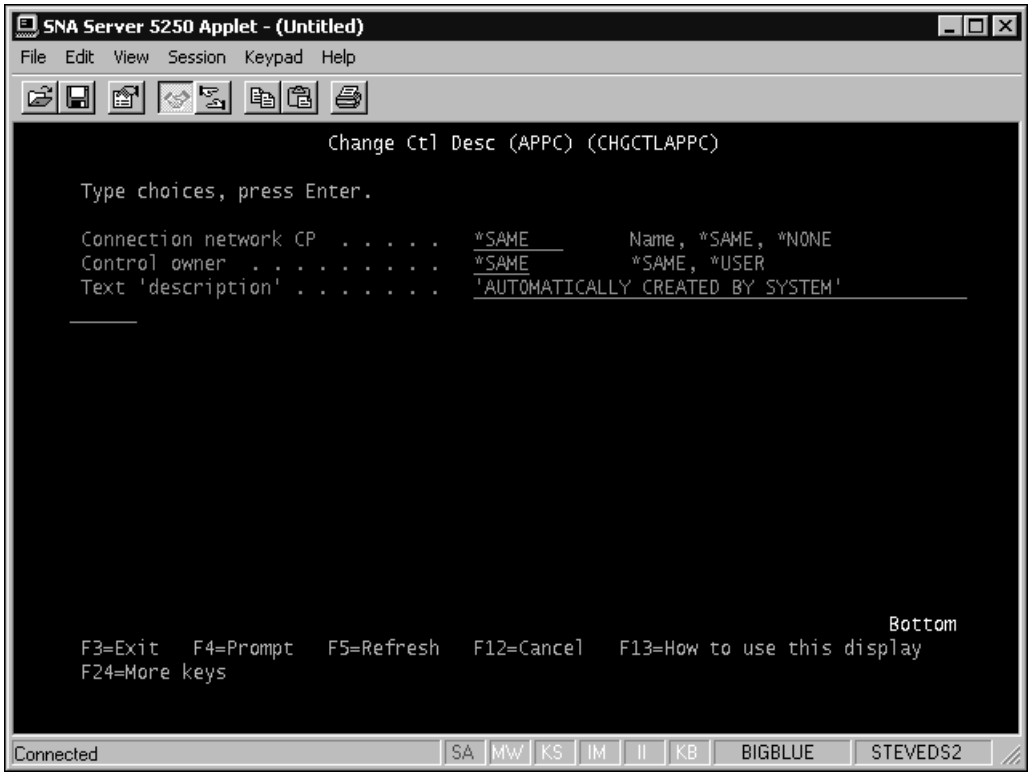
Third Change Ctl Desc (APPC) screen



Press the **PAGEDOWN** key to scroll to the fourth screen.

Here is the last of four AS/400 **Chng Ctl Desc (APPC)** screens:

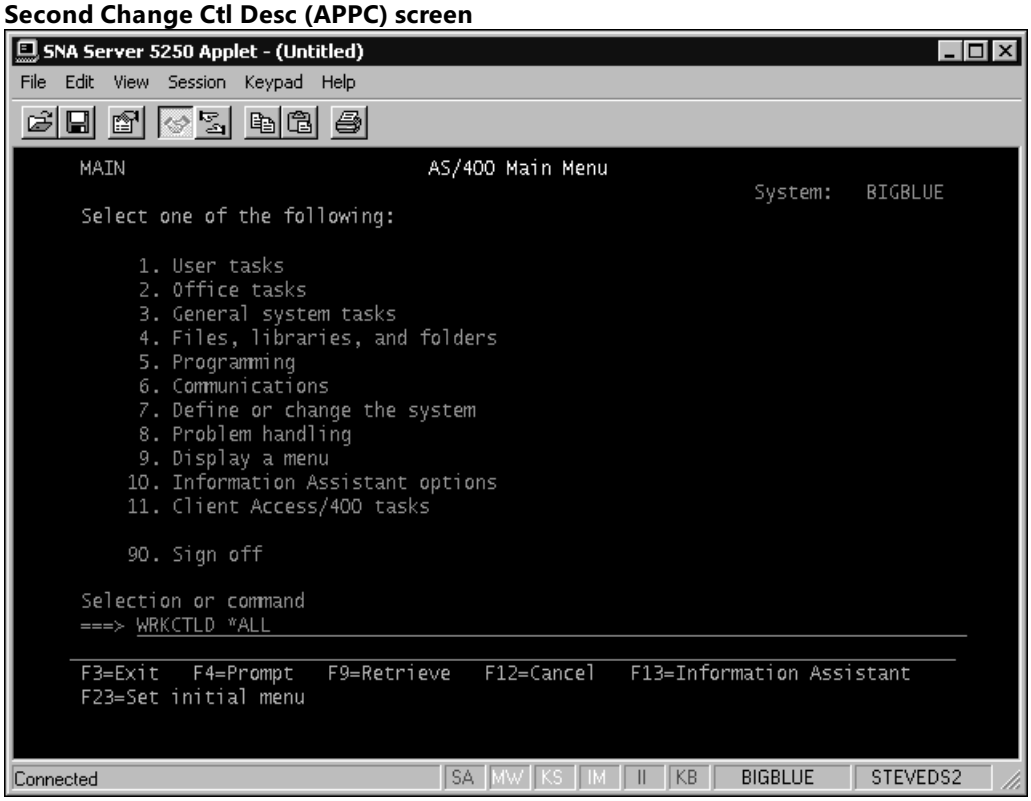
Fourth Change Ctl Desc (APPC) screen



Change Controller Status

Open the **AS/400 Main Menu**:

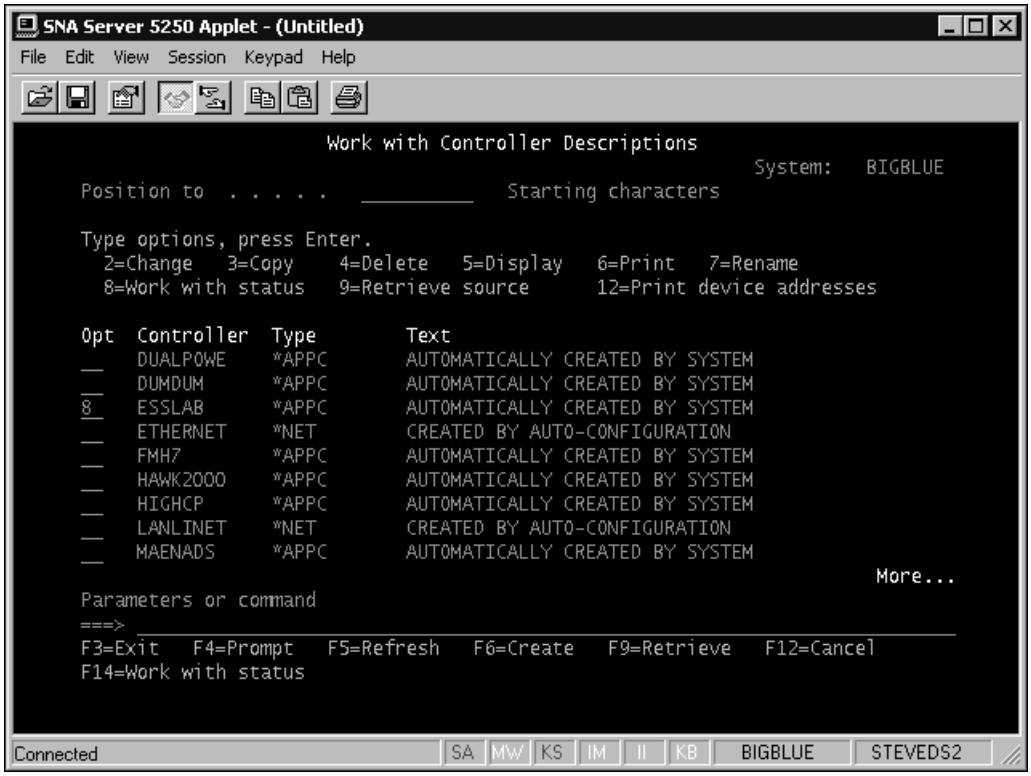
Second Change Ctl Desc (APPC) screen



Enter **WRKCTLD *ALL** at the command prompt.

This will bring up the **Work with Controller Descriptions** screen:

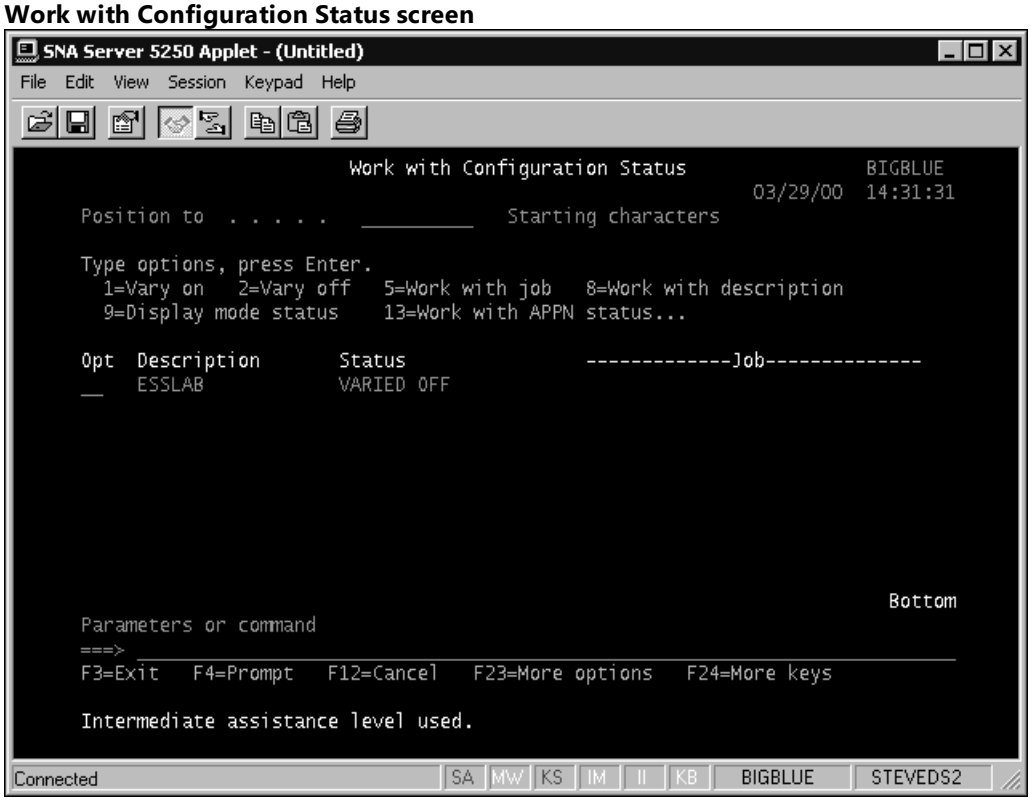
Work with Controller Descriptions screen



Choose the **Status** option for the desired controller by entering the number 8 next to the Controller that you want to change.

Here is the AS/400 **Work with Configuration Status** screen:

Work with Configuration Status screen



Select option 2 to vary the controller status off, and option 1, to vary the controller status back on.

802.2 Connection Failures

- Event ID 23
- Event ID 230
- Event ID 56
- Event ID 49

Event ID 23

Troubleshooting Mainframe and AS/400 Active to Pending DLC Connections

When a connection is Active and reverts to Pending, you should check the Event Viewer Application log for Event ID 23 and outage conditions described in the message. The qualifier reported in the Event 23 message is a DLC status code returned by the DLC interface as response to the SNA Manager 802.2 link service. This status code is interpreted by the link service and mapped to the actual "qualifier" that is reported in Event 23.

Possible causes for Event 23 Link Lost Outage Code AF and DISC Received outage code AE include LAN or router problems, timeouts or actions that need to be performed by the remote system. To diagnose bridge or router related problems we suggest using a Network Monitor to read the DLC Packets from Host Integration Server to the Host to determine the device that is causing the connection to break.

Another recommendation would be to check the Max BTU Length setting for the connection found on the **DLC 802.2** tab of the **Connection Properties** dialog box. The Max BTU is also called an I-frame; it is the number of bytes that can be transmitted in a single data-link information frame. Set the Max BTU Length so that it matches the capacity of the adapter and the host or downstream system. Otherwise, when the mainframe sends a logon screen, the BTU length that is used will be unworkable, causing the connection to drop. The MAX BTU length setting in the **Connection Properties** should equal the MAXDATA parameter in the PU definition on the mainframe or equal the MAXFRAME parameter on the AS/400. For Token-Ring Adapter Type that transmits at 4 Mbps the MAX BTU setting should be set to less than or equal to 4195. For Token-Ring Adapter that transmits at 16 Mbps, you can specify 16393. For Ethernet Adapter Type you can specify less than or equal to 1493.

With 802.2 connections, you can adjust other settings such as 802.2 Timeouts and Retry Limit settings found on the **DLC 802.2** tab of the **Connection Properties** dialog box in Host Integration Server Manager. However, the defaults for the timeouts and retries are appropriate for most networks. Knowledge Base Article Q129786 explains the details on how connection timers function and the settings to adjust if timeouts on the DLC connection are causing the connection to drop.

Other Event 23 qualifiers include:

- Outage code 29 = Remote node not active
- Outage code AB = SABME received while connection active
- Outage code AC = Frame reject sent
- Outage code AD = Frame reject received

Frame rejects occur when the receiver or sender acknowledges or detects that a DLC frame is out of sequence or invalid. We recommend taking Network Monitor traces to diagnose what intermediate device on the network is causing the frame reject to occur.

Event ID 230

This troubleshooter walks you through the process of changing a pending status to an active status for a Windows 2000 Event ID 230 connection error.

Host Integration Server generates Event ID 230 when it encounters the following conditions:

- The DLC connection to the host did not activate, and reached Pending status.

  -or-

- A connection outage (which additionally generates an Event ID 23) caused a connection to change from Active to Inactive, and finally to Pending.

Complete the following steps to troubleshoot Event ID 230:

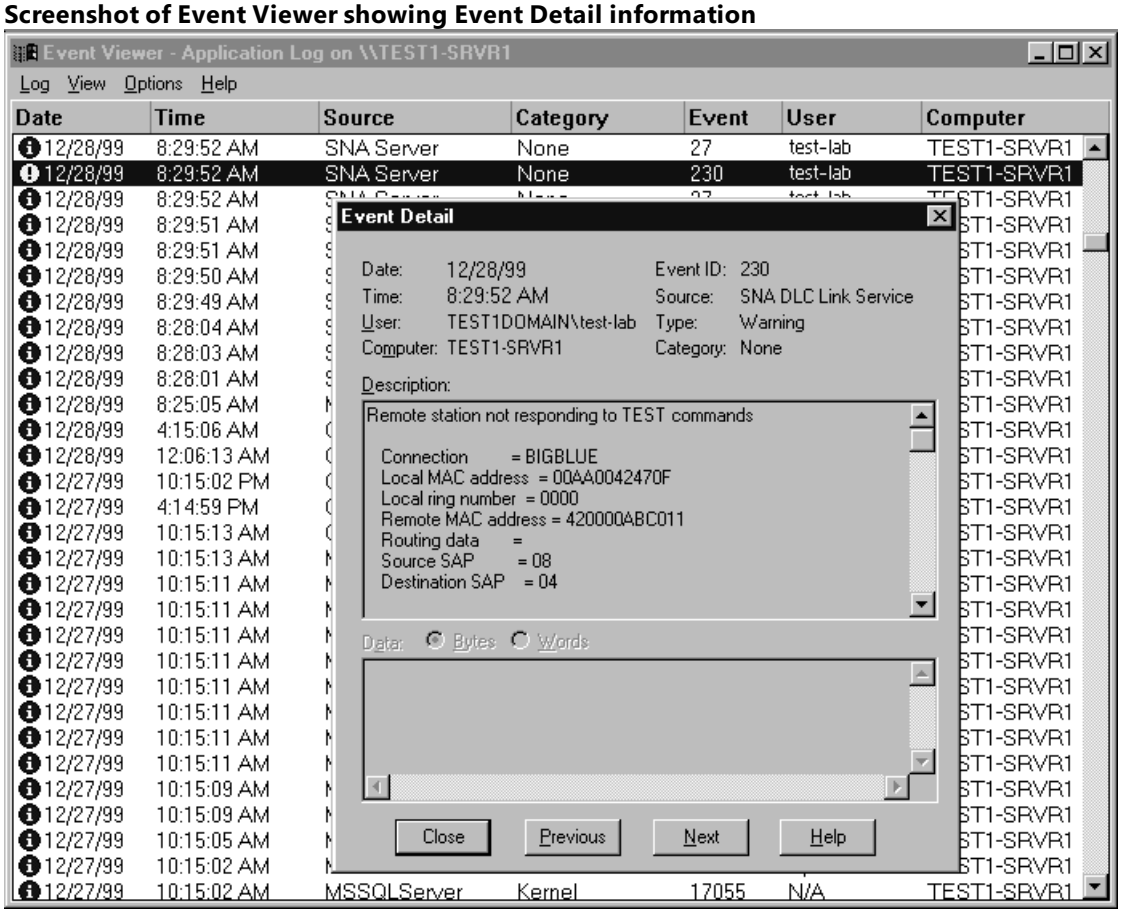
1. Review the topic Connection Initialization Sequence, especially steps 1 through 4.
2. Review the topic Finding Relevant Information.
3. Examine the Windows Application Event Log.
4. Resolve the problem: Failure Conditions and Solutions.

Windows Application Event Log

If a DLC connection fails to activate and reaches Pending status, and Host Integration Server determines that the activation failure is due to an Event ID 230 error, the following entry will be created in the Windows 2000 Application Event Log.

Note
The values displayed are for illustration purposes only.

Screenshot of Event Viewer showing Event Detail information



Failure Conditions and Solutions

There are several failure conditions that will cause an 802.2 connection to reach Pending status, with an associated Event ID 230 Application Log entry:

Failure Condition 1 — Remote Network Address mismatch

Problem: The **Remote Network Address** (Host Integration Server **Connection Property** dialog box, **Address** Tab), does not match the AS/400 **Local adapter address**.

Solution: You cannot change the AS/400 **Local adapter address** without changing its network adapter, in which case it will likely still not match. However, you can simply update the **Remote Network Address** so that it matches the AS/400 **Local adapter address**.

Failure Condition 2 — Improperly configured SNA Link Service

Problem: If the Host Integration Server computer has multiple network adapters, verify that the network adapter used to communicate with the host is properly configured with the SNA Link Service.

Failure Condition 3 — Network or Bridge Issues

Problem: Network addresses match, and SNA Link Service is properly configured on the correct network adapter, but still no response to the TEST XID command.

Solution: The connection may not be able to locate the remote network address due to Network or Bridge related issues. Try to capture a Microsoft Network Monitor or Sniffer™ trace from server to host, and determine the device that is not responding, or forwarding, the DLC Test command.

Failure Condition 4 — Remote SAP Address Mismatch

Problem: The **Remote SAP Address** on the **Address** tab of the **Connection Properties** dialog does not match the **LAN SSAP** defined in the APPC Controller Description.

Solution:

Manually edit either the AS/400 fields, or the Host Integration Server fields such that the parameters match.

Failure Condition 5 — Unable to create new APPC Controller

Problem: The AS/400 was unable to create a new APPC Controller for some reason.

Solution: If you are creating a new APPC Controller - perhaps bringing up a connection for the first time, or maybe you have tried to bring up a new APPC Controller several times, and the initialization state is now indeterminate:

1. Delete the current controller configuration on the AS/400.
2. Set the **Autocreate controller** parameter in the **AS/400 Line Description** to **Yes**.
3. Re-establish a connection, and allow the AS/400 to dynamically create a new APPC Controller.

Failure Condition 6 — Error States

Problem: The AS/400 APPC Controller or VTAM PU is in an Error State, such as Inactive or Pending Status, perhaps because there were several attempts to start the connection which were unsuccessful.

Solution: Delete the current controller configuration, set **Autocreate controller** to **Yes**, and right-click the connection, and then click **Start**.

Failure Condition 7 — Recovery Pending Status

Problem: The AS/400 line or the APPC controller is in an RCYPND (Recovery Pending) Status.

Solution:

- Check if the AS/400 line or the APPC controller is in an RCYPND (Recovery Pending) Status, when a connection is not responding to XID commands. To check for the status of the connection found in the AS/400 line description, issue the following AS/400 command:

```
" WRKCFGSTS *lin <line name> or WRKLIND <line name>
```

It is also possible to start from the AS/400 Main Menu, enter "WRKLIND *ALL", then choose the **Status** option on the line description that you want to check.

- Alternatively, you can check for an RCYPND status by issuing "WRKCTLD *ALL", then choose the **Status** option on the controller you want to check.

If the AS/400 line status is RCYPND

1. Stop the Server connection from the Host Integration Server Manager.
2. From the AS/400 WRKCFGSTS screen, choose option 2 to vary off the line status. Wait for the line to show a status of Varied Off. (Press F5 to refresh the screen).
3. Select **option 1** to Vary On the line description. Press F4 for the **Vary Configuration** dialog box. The last option in this box is **Reset**, which is "No" by default. Change to "Yes". This will reset the Ethernet Card in the AS/400. Status should then change on the AS/400 to Vary On Pending.
4. Restart the Host Integration Server Connection.

Common Connection Failure Scenarios

Scenario One:

You have a new machine and an old adapter card. This is probably the single most common cause of an Event ID 230 failure: You just upgraded your Host Integration Server computer platform and removed the C: drive. In attempting to restart a data link connection, you get an Event ID 230, "Not responding to XID commands" in your NT Event log. The AS/400 controller that you are trying to connect to is expecting to be connected with the old network adapters address. To correct this, delete the controller on the AS/400 side, (or manually change it to match your new adapter address although this is not advisable). Then, set **Autocreate Controller** on, and reconnect from the SNA side. The AS/400 will create a new controller description based upon the network adapter address of your *new* network adapter.

Scenario Two:

In a similar situation, you have a new machine and an old COM.CFG file. Your Host Integration Server computer platform is upgraded. Rather than reuse your old C drive, you reinstall the product on the new machine, and copy over the old Host Integration Server computer COM.CFG file. This causes the same problem. The old adapter address is contained in the COM.CFG file. When the AS/400 attempts to answer your NULL XID, it cannot match the parameters. The new network adapters address is contained in the NULL XID, but the AS/400 is expecting to see the old adapter address. To correct this, delete the controller on the AS/400 side, set **Autocreate Controller** on, and reconnect.

Scenario Three:

You need to restart Host Integration Server computer several times and experience problems. If the Host Integration Server computer's AS/400 connection is restarted several times in close succession (within a few minutes), then the AS/400s APPC controller may get hung in an RCYPND state. This requires manual intervention on the AS/400 to clear it.

Event ID 56

This troubleshooter walks you through the process of changing a pending status to an active status for an Event ID 56 connection error.

Host Integration Server generates Event ID 56 when it receives an XID Negotiation Error (X22) Control Vector from the host. This control vector includes an offset pointer into the XID that was sent to the host. This offset pointer points to the parameter that the host did not like.

Complete the following steps to troubleshoot Event ID 230:

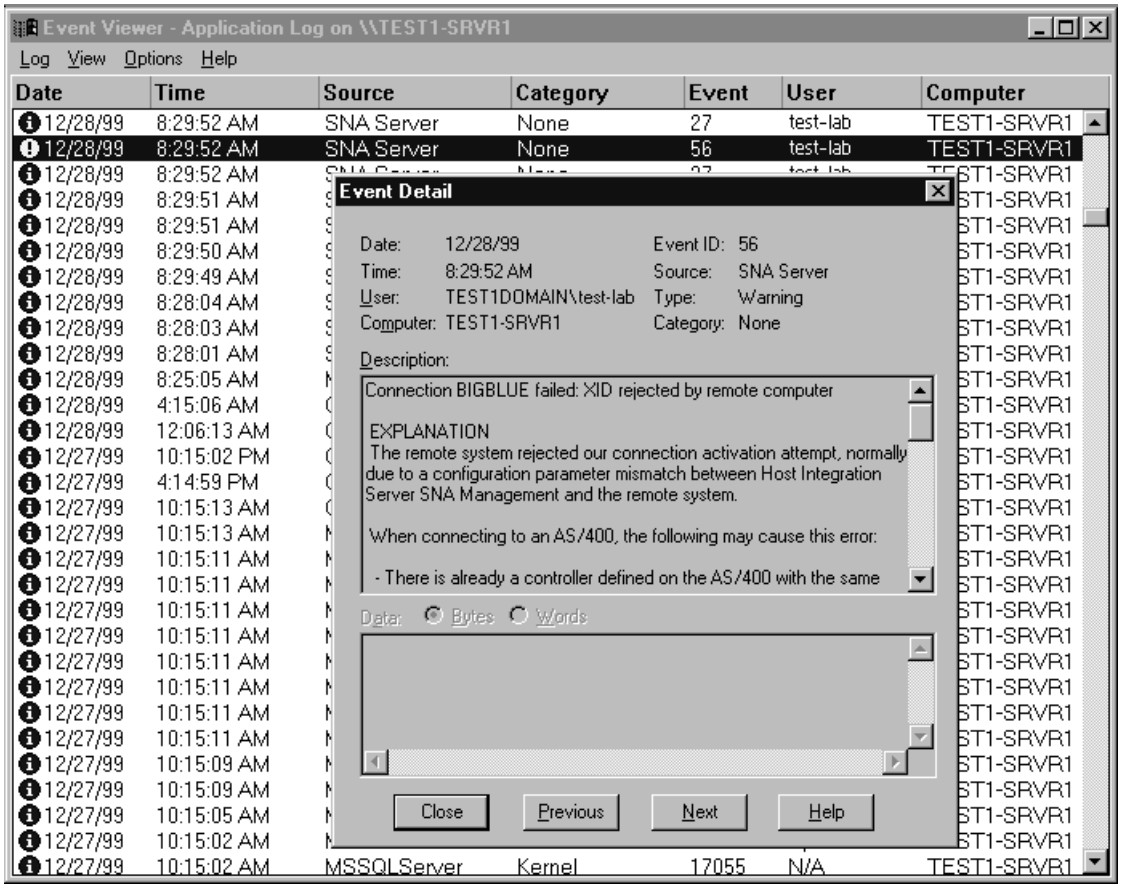
1. Review the topic Connection Initialization Sequence, especially steps 5 and 6.
2. Review the topic Finding Relevant Information.
3. Windows 2000 Application Event Log.
4. Resolve the problem: Failure Conditions and Solutions.

Windows 2000 Application Event Log

If a DLC connection fails to activate and reaches Pending status, and Host Integration Server determines that the activation failure is due to an Event ID 56 error, the following entry will be created in the Windows 2000 Application Event Log.

Note
The values displayed are for illustration purposes only.

Event Viewer with Event Detail information



Failure Conditions and Solutions

There are four failure conditions that can cause an 802.2 connection to reach Pending status, with an associated Event ID 56 Application Log entry:

Three of the failures are caused by a mismatch between a Host Integration Server configuration field, and an AS/400 field, and the last is due to an XID format mismatch.

This table summarizes the fields that must match between the Host Integration Server, and the AS/400, for a given connection:

Failure Condition	Host Integration Server field name	AS/400 Field Name
1	"Local" Network Name	Remote network identifier
2	"Local" Control Point Name	Remote control point
3	Local network adapter address (see note)	LAN remote adapter address

Failure Condition 1 — Local Network Name Mismatch

Problem: The **"Local" Network Name** on the System Identification tab of the Connection Properties dialog box does not match the **Remote network identifier** in the APPC Controller Description.

Solution: Manually update the parameter on either platform, or delete/recreate the APPC Controller, as in Failure Condition 3.

Failure Condition 2 — Local Control Point Name Mismatch

Problem: The **Local Control Point Name** on the System Identification tab of the **Connection Properties** dialog does not match the **Remote control point** defined in the APPC Controller Description.

Solution: Manually update the parameter on either platform, or delete/recreate the APPC Controller, as shown in Failure Condition 3.

Failure Condition 3 — Local Adapter Address Mismatch

Problem: The Host Integration Server computer's network adapter address (found via ipconfig /all via DOS box) does not match the AS/400 **LAN remote adapter**.

Solution: It is possible to manually edit the AS/400 **LAN remote adapter**, so that it matches the network adapter address of the Host Integration Server computer. However, it is not advisable, due to the possibility of creating parameter mismatches. Instead, perform the following procedure.

1. Delete the current controller configuration on the AS/400.

2. Set the **Autocreate controller** parameter in the AS/400 Line Description to Yes.

3. Re-establish a connection, allowing the AS/400 to dynamically create a new APPC controller that correctly binds the Host Integration Server computer's local network adapter address to the SNA Managers **"Local" Network Name**, and **"Local" Control Point Name**.

Failure Condition 4 — Invalid XID format type

1. Problem: The Host Integration Server computer is configured with XID Type: Format 0, when the AS/400 requires XID Type: Format 3.

2. Solution: Select the XID Type 3 radio button on the System Identification tab of the Connection Properties dialog.

Event ID 49

This troubleshooter walks you through the process of changing a pending status to an active status for an Event ID 49 connection error.

Host Integration Server generates Event ID 49 when it receives an XID from an AS/400 or mainframe host that contains invalid "local" parameters (as seen from the host).

Complete the following steps to troubleshoot Event ID 49:

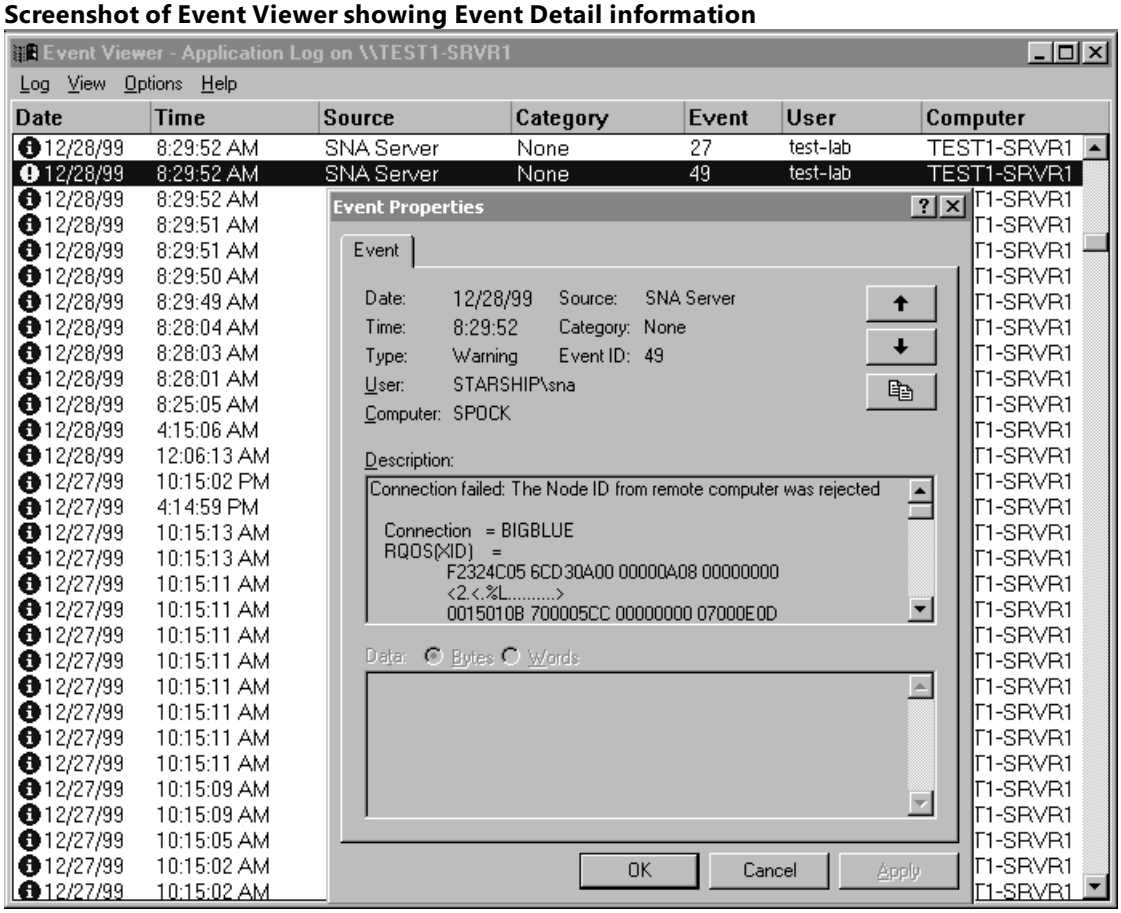
1. Review the topic Connection Initialization Sequence, especially step 7.
2. Review the topic Finding Relevant Information.
3. Examine the Windows 2000 Application Event Log.
4. Resolve the problem: Failure Conditions and Solutions.

Windows 2000 Application Event Log

If a DLC connection fails to activate and reaches Pending status, and Host Integration Server determines that the activation failure is due to an Event ID 49 error, the following entry will be created in the Windows 2000 Application Event Log.

Note
The values displayed are for illustration purposes only.

Screenshot of Event Viewer showing Event Detail information



Failure Conditions and Solutions

There are three failure conditions that can cause an 802.2 connection to reach Pending status, with an associated Event ID 49 Application Log entry:

This table summarizes the fields that must match between Host Integration Server, and the AS/400, for a given connection:

Failure Condition	Host Integration Server field name	AS/400 Field Name
1	"Local" Network Name	Remote network identifier
2	"Local" Control Point Name	Remote control point
3	Network Adapter Address	LAN remote adapter address

- These parameters are located on the Host Integration Server Connection Properties dialog box, System Identification tab, in the lower group box.

Note
The field name "Network adapter address" is not an actual field within Host Integration Server. Network adapter address refers to the address of the network adapter on the Host Integration Server computer.

Solution:

When connecting against an AS/400, these parameters should be left BLANK (i.e., empty), since they are not needed when connecting to an AS/400. However, if they were configured, and they dont match the values that the AS/400 is sending, Host Integration Server will reject the XID and log Event 49, followed by one or more Event 56s.

The main benefit that these fields provide is the ability for Host Integration Server to distinguish connection attempts by different remote systems.

Most connections are set for "outgoing". But for "incoming" connections, Host Integration Server "listens" for incoming connection attempts, and then uses the configuration information to determine who is trying to talk to it. When communicating to hosts and AS/400s, it is extremely rare for the remote system to invoke the connection. In 99% of the cases, the Host Integration Server computer establishes the connection to the remote system (i.e., an "outgoing" connection). As such, these fields should be left empty.

Step-by-Step Configuration Instructions

The following information was extracted from the Knowledge Base query "Configuring Host Integration Server computer to Talk to AS/400 Over 802.2" http://go.microsoft.com/fwlink/?LinkId=14394. This will walk you through the Host Integration Server - AS/400 connection configuration process.

This article describes how to configure a Host Integration Server computer to communicate with an AS/400 over 802.2 (Token-Ring or Ethernet), to support 5250 emulation. Both Host Integration Server and AS/400 configuration parameters are discussed.

These instructions assume the following:

- The DLC transport driver has been installed and is bound to the correct network adapter.
- The Host Integration Server computer's DLC link service has been installed using the Host Integration Server Setup program.
- The system has been restarted.

To configure the following entries using the Host Integration Server SNA Manager

1. In the **SNA Manager**, right-click **SNA Service**, and select **Properties**.
2. Enter the following parameters:

   Network Control Point Name:

   Network Name = APPN (or Remote Network ID, RMTNETID value on the AS/400).

   Control Point Name = This is the remote control point name (RMTCPNAME) value in the AS/400 APPC controller definition. For simplicity, this should be set to the local Windows 2000 computer name.

   Click **OK**.
3. Right-click **Local APPC LUs**, point to **New**, click **Local LU**, and then enter the following parameters:

LU 6.2 Type	**Independent**
LU Alias	For simplicity, this should be the same as the Local LU Name below.
Network Name	APPN (same as network name above)
LU Name	This is the remote control point name in the AS/400 APPC controller description (RMTCPNAME).

   All other entries can remain at default values.

Note
It is most efficient for all Host Integration Server computer users to use the same Local APPC LU. However, it is possible to create a unique LU for each user if desired, where the LU alias and LU name above could be replaced with the actual user name (though this requires additional administration to maintain).

   Click **OK**.
4. Right-click **Connections**, point to **New**, click **802.2**, and then enter the following parameters.

   Link Service = SnaDlc1 (or name specified during SNA setup)

   Remote End = Peer System

   Activation = If set to "On Server Startup" (default), then the Switched disconnect (SWTDSC) value on the AS/400 controller definition should be set to NO. This causes the underlying link to stay active even if there are no active sessions. Otherwise, if SWTDSC is YES, then Host Integration Server computer should be set to "On Demand" activation.

Allowed Directions = Outgoing calls. Check Incoming Calls if you want the AS/400 to activate the link.

Zoom on the Setup button:

Remote Network Address = Set to the network adapter address of the AS/400 (ADPTADR), located in the AS/400 line description.

Local Node ID = EXCHID value on the AS/400 controller description (if not specified on the AS/400, leave at default - 05D FFFFF).

Remote Node Name:

Network Name = APPN (or remote network ID RMTNETID value on the AS/400)

Control Point Name = AS/400 local control point name, configured in the AS/400 **Display Network Attributes** screen.

Remote Node ID = Not used (leave blank).

Advanced options can be left at their default settings.

Click **OK**.

5. Add a new Remote APPC LU off the connection by choosing the Insert button and enter the following parameters:

   LU Alias = For simplicity, this should be the same as the Remote APPC LU Name below.

   Network Name = APPN (or remote network id RMTNETID value on the AS/400).

   LU Name = Must be set to the AS/400 local control point name (set in the AS/400 Display Network Attributes screen).

   Uninterpreted LU Name = Not used (leave at default setting).

   Select Supports Parallel Sessions (required).

   Zoom on the Partners button. The Remote APPC LU must be partnered with the Local APPC LU created above, using the QPCSUPP mode. Because Enable Automatic Partnering is enabled by default for APPC LUs, this pairing will already be added.

   Click **OK**.

6. Save the Host Integration Server computer configuration file and restart the Host Integration Server computer service. Once the Host Integration Server computer service is Active, all connections configured to activate on server startup will go into a Pending state, then switch to Active.

Note
If the connection stays in Pending mode and doesn't activate:

7. Check the Windows 2000 Application Event log (using Event Viewer) to see the reason why the connection is not activating. For example, if the AS/400 is not responding to TEST commands, then the remote network address may not be correct.
8. Double check the configuration entries above with your AS/400 system administrator and make sure the AS/400 line is active. If all appears okay, check the Host Integration Server computer controller description on the AS/400. For LAN-based connections (such as Token-Ring or Ethernet), the AS/400 defaults to auto configuration for new controllers, so manual generation on the AS/400 is not required.
9. Check to see if the AS/400 is logging any errors when Host Integration Server computer attempts to establish the connection.
10. If the connection activates but users are unable to open 5250 sessions, zoom on the Status button on the Local APPC LU, which should show the following (when working correctly):

Partner LU	**Mode**	**User Name**	**Client**	**Limit**
<remote LU>	SNASVCMG	<User Name>	<Client Name>	2
<remote LU>	QPCSUPP	<User Name>	<Client Name>	64

This means that up to 64 sessions are available for use, though no sessions are currently active.

11. At this point, a 5250 user can then open any valid APPC LU/LU pair supported by any Host Integration Server computer in the domain. However, in order to simplify 5250 user access through Host Integration Server, there are various options available to a Host Integration Server computer administrator and the 5250 user, including the following:

    - Default AS/400 session for a user: On the Host Integration Server computer user/group record, the administrator can define default local and remote APPC LUs. If defaults are configured, and the 5250 client is used, this is the LU/LU pair the 5250 user will open when the 5250 client local and partner APPC LU names are left blank.

    - If default APPC LUs are not preassigned to a user/group, here is how to configure pools of Local APPC LUs and Remote (partner) APPC LUs to simplify 5250 user access to one or more AS/400's supported by Host Integration Server.

    - Local APPC LU pool: Define the Local APPC LU as a "Member of Default Outgoing Local APPC LU Pool". If a 5250 user does not enter a Local APPC LU when opening a session, the user will access one of the available Local LUs in this pool.

    - Remote APPC LU pool: The Remote APPC LU pool is determined by the Remote LUs that are partnered with the Local APPC LU and the QPCSUPP mode. So, if a Local APPC LU is specified, but the Remote APPC LU is left blank, a 5250 session will be allocated from an available Remote APPC LU with which it is partnered.

Sample AS/400 Configuration

The following AS/400 configuration screens are shown below, along with their corresponding configuration setting for SNA. This includes the AS/400 network attributes screen, Token-Ring line description, APPC controller, and virtual device description.

Many of the configuration settings are not relevant for communications to function, though are shown here for completeness.

AS/400: Display Network Attributes screen

Current system name	BIGBLUE
Pending system name	
Local network ID	APPN (must match the remote network name on the Host Integration Server computer connection)
Local control point name	BIGBLUE (must match the remote network name on the Host Integration Server computer connection)
Default local location	BIGBLUE
Default mode	BLANK
APPN node type	*ENDNODE
Maximum number of intermediate sessions	200
Route addition resistance	128
Server network ID/control point name	APPN BLUE
Alert status	*OFF
Alert logging status	*NONE
Alert primary focal point	*NO
Alert default focal point	*NO
Alert backup focal point	
Network ID	*NONE
Alert focal point to request	
Network ID	*NONE
Alert controller description	*NONE
Alert hold count	0
Alert filter	: *NONE

Library	
Message queue	QSYSOPR
Library	QSYS
Output queue	QPRINT
Library	QGPL
Job action	*FILE
Maximum hop count	16
DDM request access	*OBJAUT
PC Support request access	*OBJAUT
Default ISDN network type	
Default ISDN connection list	QDCCNNLANY

AS/400: Change Line Desc (Token-Ring) (CHGLINTRN), screen 1

Line description	LIND > LANLINE
Resource name	RSRCNAME LIN031
Online at IPL	ONLINE *YES
Vary on wait	VRYWAIT *NOWAIT
Maximum controllers	MAXCTL 64
Line speed	LINESPEED 16M
Maximum frame size	MAXFRAME 1033
Activate LAN Manager	*YES
TRLAN manager logging level	TRNLOGLVL *OFF
TRLAN manager mode	TRNMGRMODE *OBSERVING
Log configuration changes	LOGCFGCHG *LOG
Token-Ring inform of beacon	TRNINFBCN *YES
Local adapter address	ADPTADR 494061026052
Functional address + for more values	FCNADR *SAME

Note

The **Local adapter address** must match the remote network address on the Host Integration Server computer connection.

Also, the Local Adapter Address (ADPTADR) parameter on the line description (for Token-Ring or Ethernet) has a default of *ADPT. AS/400 Administrators can choose to enter their own locally administered address or the *ADPT inserts the burned-in adapter card address. The following information includes advantages of both options.

Locally Administered Address:

Advantages: The address is permanent and will not change. It is important to pick a unique address so that there are no network conflicts. This is the IBM recommended option.

***ADPT Address:**

Advantages: The adapter address is inserted into the line description when the line is varied on. Afterward, you can display the line description and see the adapter card address. This address is a unique address and you are insured there will be no duplicates on your network.

Disadvantages: If the Adapter card has a hardware problem and is replaced, you need to have a new address when the line is varied on for the first time. All users who connect using the address will not be able to communicate unless they change their configuration accordingly. This can cause problems if you have many users because they all reference the same adapter address.

AS/400: Change Line Desc (Token-Ring) (CHGLINTRN), screen 2

Source Service Access Point	04
SSAP maximum frame	*MAXFRAME
SSAP type for more values	*SNA
Early token release	ELYTKNRLS *YES
Error threshold level	THRESHOLD *OFF
Link speed	LINKSPEED 16M
Cost/connect time	COSTCNN 0
Cost/byte	COSTBYTE 0
Security for line	SECURITY *NONSECURE
Propagation delay	PRPDLY *LAN
User-defined 1	USRDFN1 128
User-defined 2	USRDFN2 128
User-defined 3	USRDFN3 128
Autocreate controller	AUTOCRTCTL *YES

Note
For 16-MB Token-Ring, note the **Early Token Release** setting. This should match the early token release setting used on you r local network adapter (set using the **Network** program in **Control Panel**).

Autocreate controller: This setting controls whether the APPC controller definition and virtual device definition (listed below)

are automatically created by the AS/400 on this line. For Token-Ring or Ethernet lines, this is the default.

AS/400: Change Line Desc (Token-Ring) (CHGLINTRN), screen 3

Autodelete controller	AUTODLTCTL 1440
Recovery limits:	CMNRCYLMT
Count limit	2
Time interval	5
Text 'description'	TEXT 'LAN Line description'

Note

If AUTOCRTCTL (Autocreate controller) is YES (in the line description above), then the APPC controller and virtual device definitions below do not have to be generated. In this case, the Host Integration Server computer configuration settings noted below will cause these AS/400 configuration entries to be automatically generated.

AS/400: Change Ctl Desc (APPC) (CHGCTLAPPC), screen 1

Controller description	CTLD	> TRUTH00
Online at IPL	ONLINE	*NO
APPN-capable	APPN	*YES
Switched line list + for more values	SWTLINLST	LANLINE
Character code	CODE	*EBCDIC
Maximum frame size	MAXFRAME	16393
Remote network identifier	RMTNETID	APPN
Remote control point	RMTCPNAME	TRUTH
Exchange identifier	EXCHID	05600001
SSCP identifier	SSCPID	*SAME
Initial connection	INLCNN	*DIAL
Dial initiation	DIALINIT	*LINKTYPE
Switched disconnect	SWTDSC	*YES
Disconnect timer:	DSCTMR	
Minimum connect timer	170	
Disconnection delay timer	0	

Note

Remote network identifierMust match the network name configured on Host Integration Server computer (set using Admin when zooming on the server name), as well as the network name configured on the Local APPC LU.

Remote control point Must match the local control point name configured on Host Integration Server computer (set using Admin when zooming on the server name), as well as the LU name of the Local APPC LU.

Exchange identifier Must match the Local Node ID in the Host Integration Server computer connection.

Switched disconnect If this is set to YES, the AS/400 drops the link when there are no active users, so the Host Integration Server computer connection should be configured to activate On Demand. Note that Host Integration Server computer 802.2 connections default to activate On Server Startup. If this is desired, the SWTDSC setting should be set to NO.

AS/400: Change Ctl Desc (APPC) (CHGCTLAPPC), screen 2

LAN remote adapter address	ADPTADR	10005A38374C
LAN DSAP	DSAP	04
LAN SSAP	SSAP	04
LAN frame retry	LANFRMRTY	*CALC
LAN connection retry	LANCNNRTY	*CALC
LAN response timer	LANRSPTMR	*CALC
LAN connection timer	LANCNNTMR	*CALC
LAN acknowledgment timer	LANACKTMR	*CALC
LAN inactivity timer	LANINACTMR	*CALC
LAN acknowledgment frequency	LANACKFRQ	*CALC
LAN max outstanding frames	LANMAXOUT	*CALC
LAN access priority	LANACCPTY	*CALC
LAN window step	LANWDWSTP *	*NONE
APPN CP session support	CPSSN	*YES
APPN node type	NODETYPE *	*CALC
APPN transmission group number	TMSGRPNBR	*CALC
APPN HPR capable		*YES

Note
LAN remote adapter addressThis is the network adapter address of the network card being used on the Host Integration Server computer.

LAN DSAP This must match the SAP address on the Host Integration Server computer link service, configured using Host Integration Server computer setup. This is usually set to 04, so this rarely (if ever) changes.

LAN SSAP This is the SAP address of the destination AS/400 computer, and must match the SAP address on the Host

Integration Server computer connection, configured using Host Integration Server computer Admin. Again, this rarely (if ever) changes. AS/400: Change **Ctl Desc (APPC) (CHGCTLAPPC)**, screen 3

APPN minimum switched status	MINSWTSTS	*VRYONPND
Autodelete device	AUTODLTDEV	1440
User-defined 1	USRDFN1	*LIND
User-defined 2	USRDFN2	*LIND
User-defined 3	USRDFN3	*LIND
Recovery limits:	CMNRCYLMT	
Count limit	2	
Time interval	5	
Model controller description	MDLCTL	*NO
Connection network ID	CNNNETID	*SAME
Connection network CP	CNNCPNAME	*SAME
Control owner	CTLOWN	*SAME
Text 'description'	TEXT	AUTOMATICALLY CREATED BY QLUS

AS/400: Change Device Desc (APPC) (CHGDEVAPPC)

Device description	DEVD	> TRUTH03
Online at IPL	ONLINE	*NO
Mode + for more values	MODE	*NETATR
Message queue	MSGQ	QSYSOPR
Library	*LIBL	
Local location address	LOCADR	00
Single session:	SNGSSN	
Single session capable	*NO	
Number of conversations		
Locally controlled session	LCLCTLSSN	*SAME
Pre-established session	PREESTSSN	*SAME
Text 'description'	TEXT	AUTOMATICALLY CREATED BY QLUS

Mode description	MODD	> QPCSUPP
Class-of-service	. COS	#CONNECT
Maximum sessions	MAXSSN	64
Maximum conversations	MAXCNV	64
Locally controlled sessions	LCLCTLSSN	16
Pre-established sessions	PREESTSSN	0
Inbound pacing value	INPACING	7
Outbound pacing value	OUTPACING	7
Maximum length of request unit	MAXLENRU	*CALC
Text 'description'	TEXT	created by <name>

Note

Device descriptionThis device name corresponds with the Host Integration Server computer Local APPC LU name (in this cas e, TRUTH), plus a two-digit device number, generated by the AS/400 (but not configured in Host Integration Server computer).

Mode description - This should not be changed; this is the default mode name for IBM PC Support clients (and Host Integration Server computer).

Maximum sessions - This is the number of sessions supported by the QPCSUPP mode, and should match the Host Integration Server computer session limit configured on the QPCSUPP mode entry, configured using the Host Integration Server computer setup program (zoom on Local or Remote LU, zoom on **Partners** button, then choose the **Modes** button).
Maximum conversations - This should match the **Maximum sessions** setting.

Solution: Vary off, then on, the controller, as previously outlined.

Troubleshooting Mainframe Pending DLC Connections - Event ID 230

If a DLC connection to the mainframe does not activate and reaches "Pending" status, you should view the Event Viewer Application Log for errors. Event 230 is an event in the application log that occurs when a connection is not responding to XID commands. We recommend checking the identifiers and/or the network address used by the host match with the corresponding parameters in Host Integration Server Manager. The identifiers or addresses must match in order for the exchange of identifiers (XIDs) and test frames to complete successfully.

The Remote Network Address (which can be viewed on the Address tab of the Connection Properties dialog box); should match the 12-digit hexadecimal network address of the remote host, peer, or downstream system. The MACADDR parameter in the VTAM PORT definition, or in the NCP Gen must match the Remote Network Address configured in the Host Integration Server Connection.

Node ID is the identifier used when exchanging identification (XIDs) with most mainframes. Check to make sure that the Node ID matches. If they do not, the Host Integration Server is not identifying itself in a way that the host can recognize. The IDBLK and IDNUM host parameters are located in the PU definition and must match the Local Node ID configured in the System Identification Tab in the Server Connection properties. We also recommend checking to see that the PU in VTAM is active.

There are some situations in which the mainframe does not use Node ID in XIDs, but instead uses Network Name and Control Point Name. These situations include mainframes communicating through LU 6.2, and mainframes that call up the Host Integration Server computer (meaning that the SNA Server computer accepts incoming calls on that mainframe connection). In these situations, the following parameters must match:

- The NETID in the VTAM Start command for the local SSCPNAME configured in the PU definition must match the Local Network Name configured on the Server. The CPNAME in the PU definition must match the Local Control Point Name configured on the server. The NETID and SSCPNAME in the VTAM Start command must match the Remote Network Name and Remote Control Point Name configured in the Connection Properties.

If all connection settings above match the corresponding host parameters, then the connection may not be able to locate the remote network address due to Network or Bridge related issues. When the DLC connection first tries to activate, it sends out an LLC TEST frame to the remote network address to initiate the link level connection. If the Server does not receive a reply to the local TEST command within 0.5 seconds, it sends an all-routes broadcast TEST command. The link service will send a total of three all-routes broadcast TEST commands if the remote station it not responding.

On a Token-Ring network, the local ring is tried first. If there is no response to the TEST frame, the Server resends the TEST frame with the "all routes broadcast" setting enabled which is then forwarded by source routing bridges.

In an Ethernet network, the Test command does not contain any source-routing information. The Server sends TEST and XID frames to both the configured remote network address and the bit-flipped address, in both DIX and 802.3 formats.

If the remote station does not reply to the TEST frame then the connection remains in a pending condition and an Event 230 gets logged in the Event Viewer Application log.

To diagnose the cause of the Test command failing we recommend capturing a Microsoft Network Monitor or SnifferTM trace from Server to host and determine which device is not responding or forwarding the DLC Test command.

Troubleshooting AS/400 Pending SDLC Connections

If the SDLC connection to the mainframe does not activate and reaches "Pending" status shown in Host Integration Server, you should view the Event Viewer Application and System Log for SDLC Link Service and Adapter errors and to check the modem light indicators. We also suggest enabling detailed Problem Analysis found on the Error/Audit Logging tab in the Subdomain properties dialog box.

If a switched connection failure occurs, we recommend checking the identifiers used by the host match to the one used by the Host Integration Server connection. The Local Node ID found on the System Identification Tab of the Connection Properties needs to match the EXCHID value on the AS/400 controller description.

When these conditions occur, the Event Viewer Application Log will contain Event ID 182 if an IBM SDLC Link Service is used.

Connection pending conditions can also occur if the AS/400 APPC controller does not match the Local Control Point Name in Host Integration Server. We recommend comparing the controller value found on the RMTCPNAME field in the AS/400 controller definition with the Local Control Point Name located on the System Identification tab in the Host Integration Server connection properties. The Network Name in the connection properties also must match the setting found in the RMTNETID value on the AS/400.

We suggest checking to see if the Encoding (NRZ/NRZI) setting in the AS400 line description matches the setting on the Address Tab in the connection properties dialog box. If the NRZ/NRZI settings do not match, the two ends of the connection begin negotiating but cannot interpret each others signals correctly. The Poll Address found on the Address Tab in the connection properties dialog box, should also be verified to match the Station address (STNADR) setting on the AS/400 controller definition. The duplex setting on the AS/400 line description must also match the Duplex setting on the SDLC tab of the connection properties in Host Integration Server Manager.

If the modem "receive data" light flashes, but "transmit data" light does not and the connection status reaches "Pending" and then the connection is dropped, we recommend checking the cables to verify that the correct V.35 or RS232 connector is used for the adapter installed in the machine. If the SDLC connection is a leased line and is a multi-drop line, we suggest checking that all devices sharing the line are set for half-duplex and that Constant RTS should be disabled.

We also suggest checking the DMA channel setting on the adapter if supported and compare this to the setting in the link service properties dialog box. If this setting specifies a nonexistent address, data cannot flow. The modem configuration should also be checked to make sure that its configured to use Synchronous instead of Asynchronous communications.

If problem persists, we recommend capturing a synchronous line trace using a third party Line Monitor and an SDLC Link Service Trace to diagnose the cause of the SDLC connection failing. The Host Integration Server Trace Program can be found on the Tools menu in Host Integration Server Manager. Select the Link Service Properties and enable data link control and Level 2 Messages from the Message Trace Tab and enable all Internal traces from the Internal Trace Tab. To analyze the traces, we suggest that you open an incident with Microsoft and send the traces to the Support Specialist.

Troubleshooting Mainframe Pending SDLC Connections

If the SDLC connection to the mainframe does not activate and reaches "Pending" status shown in Host Integration Server, it is recommended to view the Event Viewer Application and System Log for SDLC Link Service and Adapter errors and to check the modem light indicators. We also suggest enabling detailed Problem Analysis found on the Error/Audit Logging tab in the Subdomain properties dialog box.

If a switched connection failure occurs, we recommend checking to ensure the identifiers used by the host match the one used by the Host Integration Server connection. The Local Node ID found on the System Identification Tab of the Connection Properties needs to match the IDBLK and IDNUM host parameters located in the PU definition. The connection failure can also occur if the XID is already in use. We recommend verifying that the Local Node ID is not already in use and that the PU is enabled in VTAM. When these conditions occur, the Event Viewer Application Log will contain Event ID 182 if an IBM SDLC Link Service is used.

We suggest checking the Encoding (NRZ/NRZI) setting used by the host and compare this to the setting on the Address Tab in the connection properties dialog box. The NRZI setting on the Mainframe can be found in the LINE/GROUP definition and it defaults to YES. If the NRZ/NRZI settings do not match, the two ends of the connection begin negotiating but cannot interpret each others signals correctly. The Poll Address found on the Address tab in the Connection Properties dialog box, should also be verified to match the ADDR parameter found in the PU definition.

If the modem "receive data" light flashes, but "transmit data" light does not and the connection status reaches "Pending" and then the connection is dropped, we recommend checking the cables to verify that the correct V.35 or RS232 connector is used for the adapter installed in the machine. If the SDLC connection is a leased line and is a multi-drop line, we suggest checking that all devices sharing the line are set for half-duplex and that Constant RTS should be disabled.

We also suggest checking the DMA channel setting on the adapter if supported and to compare this to the setting in the link service properties dialog box. If this setting specifies a nonexistent address, data cannot flow. The modem configuration should also be checked to make sure that its configured to use Synchronous instead of Asynchronous communications.

If problem persists, we recommend capturing a synchronous line trace using a third party Line Monitor and an SDLC Link Service Trace to diagnose the cause of the SDLC connection failing. The Host Integration Server Trace Program can be found on the **Tools** menu in SNA Manager. Select the Link Service Properties and enable data link control and Level 2 Messages from the Message Trace tab and enable all Internal traces from the Internal Trace tab. To analyze the traces, we suggest that you open an incident with Microsoft and send the traces to the Support Specialist.

Performance Problems

In This Section

Maximizing Communications Performance

Servers used primarily for communications need to provide fast network performance, but do not need to provide fast file access (such as a server used primarily as a file server). Faster network communication can be achieved if portions of memory are set aside for communications by configuring "nonpaged memory". Nonpaged memory is portions of memory that are never swapped to disk, but remain available for immediate use at all times.

In Windows 2000, you can view or change network performance options, as described in the following procedures. For a Host Integration Server computer, you may not need to change this option, because Host Integration Server Setup automatically sets the option to maximize throughput for network applications.

To view or change network communication options for a Windows 2000 Server

1. Click **Start**, point to **Settings**, click **Control Panel**, and then double-click **Network and Dial-up Connections**.
2. Double-click **Local Area Connection** and click **Properties**.
3. Click **File and Printer Sharing for Microsoft Network**, and click **Properties**.
4. For a server dedicated to communications, select **Maximize data throughput for network applications**.
5. Click **OK**, and then click **OK** again. Click **Close**.

If you made a change, the **Network Settings Change** message box appears. To put the changes into effect right away, restart the computer. To put the changes into effect later (at next startup), do not restart the computer now.

Maximizing Background Processing on Host Integration Server Computers

Servers used primarily for communications run many important background processes (processes not related to user actions in the current window). These servers generally do not need to run foreground processes at maximum speed. Therefore, Host Integration Server throughput can be increased by making the operating system more responsive to background processes and less responsive to foreground processes.

As would be expected, a server that is less responsive to foreground processes will run local applications such as word-processing software, spreadsheets, or SNA Manager more slowly. Tasking is most appropriate for servers used primarily to support clients, not servers used locally as desktop computers.

To optimize background processing for a Windows 2000 server

1. Click Start, point to **Settings**, click **Control Panel**, and then double-click **System**.
2. Select the **Advanced** tab.
3. In the **Performance** box, click **Performance Options**.
4. In the **Application response** box,select **Background services**.
5. Click **OK**, and then click **OK** again.

Host Print Service Problems

In This Section

Common Problems

The Host Print Service runs under a Windows 2000 user context. The Host Print Service (SnaPrint) may not be running under a Windows 2000 user context that has authority to open a session to the destination printer. Confirm the user context of the Host Print Service, and/or try re-entering the password within the Services Control Panel for SnaPrint Startup, within "Log On As: This Account:". In addition the user's rights can be confirmed by logging on to Host Integration Server as that user and attempting to print from Notepad.

The Generic/Text only printer driver has several limitations. Please see the following Microsoft Knowledge Base articles (available from the Web site http://go.microsoft.com/fwlink/?LinkID=99570):

168233: http://support.microsoft.com/kb/168233

166000: http://support.microsoft.com/kb/166000

162616: http://support.microsoft.com/kb/162616

154322: http://support.microsoft.com/kb/154322

Transparent sections in print jobs will be discarded unless a PDT file is configured. When such sections are discarded, an Even is logged in the Event Log, with a sample of the discarded data. This logging can be disabled globally from the Properties Page of the Print Server in SNA Manager.

HP PCL escape sequences in transparent sections should use all ASCII characters.

In addition the session property **Transparent is ASCII** must be selected.

If printing through cascaded Windows 2000 print servers, it may be necessary to add the printer share name to the NullSessionShares Registry entry on the Windows 2000 computer to which the Host Print Service is printing:

```
HKEY_LOCAL_MACHINE
   System/CurrentControlSet
     Services
       LanmanServer
          Parameters
             NullSessionShares:  REG_MULTI_SZ:  <sharename>
```

Where <sharename> is the share name associated with the Windows printer. Note that each share name must be listed on a separate line within the REGEDT32 "Multi-String editor". The Windows 2000 computer must be rebooted to enable this change. See Microsoft Knowledge Base article 121853 at: http://support.microsoft.com/kb/121853.

Print Tracing Problems

Host Integration Server incorporates extensive tracing within the Host Print Service components and the messages that flow between them. The Host Integration Server Print Service communicates over two well-defined SNA APIs - FMI (Functional Management Interface) and the APPC (Application Program to Program Communication) API FMI is used for 3270 printing, while APPC API is used for AS/400 printing.

For problems where the output is not correct

1. If possible, isolate this print job by stopping all printing to other printer sessions. This will make it easier for support personnel to analyze when viewing traces.
2. Stop the print session(s) in question.
3. Enable the following traces using the SNA Manager Trace Utility:
4. Select **SNAPrint**: Internal Trace Tab (Custom Events), Message Trace (all). Custom Events enables a new type of tracing called Advanced Job Logging. It traces each byte of the data stream.

   Select **SNAServer**. Message Trace (Data Link Control, 3270 Messages, LU 6.2 Messages).

   Reproduce the problem.

   Turn the traces off **immediately** by selecting Clear All Traces button in the Tracing Items Tab.

Print another job to this Print Session, this time changing the Destination to File. This can be done in the Printing Tab for this Print Session.

For all other problems

1. If the problem manifests itself with only one print job, isolate this job by stopping all printing to other print sessions, if possible. This will make it easier for support personnel to analyze when viewing traces.
2. Stop the print session(s) in question.
3. Enable the following traces using the SNA Manager Trace Utility:

   Select **SNAPrint**: Internal Trace (all; with the exception of Custom Events), Message Trace (all).

   Select **SNAServer**. Message Trace (Data Link Control, 3270 Messages, LU 6.2 Messages).

   Reproduce the problem.

   Turn the traces off immediately by selecting **Clear All Traces** in the **Tracing Items** tab.

Collecting Information for support personnel

1. %snaroot%\system\config\com.cfg.
2. All traces in the %snaroot%\traces directory.
3. Application Event log.
4. Printer output file in cases where the problem is wrong output.
5. Hard copy output from the physical printer.
6. An output file and / or a hard copy output of a "successful" job. This might be another print emulator or an output from an actual IBM device.

Note
Before contacting product support, obtain the following information:

3270 Printing Problems

The 3270 datastream was not designed for proportional fonts. This can cause problems in some print jobs, resulting in characters that overlap. The advanced settings of the Print Server Properties page allows you to configure Host Print Service to use a different method of positioning characters.

Problems with Form Feeds

One commonly seen problem with Host Print Service is extra or missing form feeds (FF). Some of these issues involve how SNA Print handles explicit form feeds. Other issues relate to using the number of lines per page, in place of a FF character, to cause a page break (form feed).

When Host Print Service receives a FF character in the host data stream ('0x0C'), it holds this character until it receives additional data, either control codes (SCS or 3270 orders) or printable characters. If it receives additional data, the FF is sent to the printer and the additional data is processed. If no further data is received, meaning we are at the end of the job, the FF is dropped. At this point the SNA Print will complete the outstanding job by calling either **EndDoc** for sessions not using a PDT or **EndDocPrinter**, for sessions using a PDT. When **EndDoc** is called, a FF is added to the end of the job. When **EndDocPrinter** is called, no FF is added. In this latter case, whether SNA Print adds a FF to the end of the job depends on how the END_JOB parameter is configured in the PDT. An alternative to using the PDT is to change the default data type for the print processor in the Windows NT Printer properties. If the default data type is set to RAW [auto FF], the print driver checks for the presence of a FF and adds one if necessary.

It is possible to force SNA Print to not drop the final FF when using a PDT. This requires the Registry entry FlushFF be added and set to TRUE.

```
FlushFF:  REG_SZ
   HKEY_LOCAL_MACHINE
     SYSTEM
       CurrentControlSet
          Services
             SnaPrint
               Parameters
```

FF at End of Job	PDT	FF added	End results
Yes	No	Yes	FF
No	No	Yes	FF
Yes	Yes	No	(depends on PDT)
No	Yes	No	(depends on PDT)

Many older host print jobs rely on the number of lines per page to determine page breaks. They assume for example that a job will use 66 lines per page, so add enough blank lines after the text to bring the total number of lines to 66 before starting the text that should be on the next page. If there were 30 lines of text, 36 blank lines would be added before the text intended for the next page. The drawback of this method is it depends on the printable area of the printer, the lines per inch, the lines per page, and the top margin set for the job. If by default only 65 lines will fit per page, the resulting printout will show "page creep," where the last blank line is pushed to the top of the next page, and then two lines to the top of the third page, etc. This "page creep" can be remedied within the PDT file by having the START_JOB parameter set the top margin to zero and the lines per page to 66. In addition the Printer Session properties should have the lines per inch set to 6.

For example with a printer using HP PCL the following would be added to the PDF:

In the macros section:

```
TOP    EQU 1B 26 6C 30 45    /* Top Margin set to 0 */
STL    EQU 1B 26 6C 36 36 46        /* Set Text Length to 66 */
```

For Start Job

```
START_JOB = TOP0 STL
```

Host Print Service is designed to execute a form feed (FF) included in an LU 3 print job when any of the following conditions are met:

- If the FF is inserted as the first character after the WCC in a 3270 Erase/Write or Erase/Write Alternate command.
- If the FF is located after a valid NL (New Line) order.
- If the FF is located after the last printable character position of any print line.

A registry entry is available that will force Host Print Service to honor all form feed characters in an LU 3 print job, even if they do not meet the above conditions. To add this entry, find the following key using REGEDT32:

```
HKEY_LOCAL_MACHINE
   SYSTEM
    CurrentControlSet
      Services
        SnaPrint
          Parameters
```

Add the following entry to this key:

```
Value Name:
Data Type:
String:
```

DoAllLU3FFs should be set to **TRUE**. The system checks to see if this registry entry exists. Any value entered for the string will enable this feature.

Non-printable Character Problems

Characters below 0x40 in the 3270 datastream are considered "non-printable" characters. Some of these values are used by 3270 Orders and SCS codes. By default, if Host Print Service encounters a non-printable character that is not an SCS code or 3270 Order, it rejects the frame, or in some cases translates the character to an ASCII space and continues on. To force Host Print Service to process all characters, and translate them to their ASCII equivalents according to the code page being used, a registry entry has been made available. To add this entry, find the following key using REGEDT32:

```
HKEY_LOCAL_MACHINE
   SYSTEM
     CurrentControlSet
       Services
         SnaPrint
           Parameters
```

Add the following value to this key:

```
Value Name:  HonorCharsUnder0x40
Data Type:  REG_SZ
String:  TRUE
```

Configuration Problems

In This Section

Config Lock and Out-of-Date Messages in the Status Bar

The SNA Manager will only lock the configuration file when you initiate a configuration change. If the lock is obtained, the status bar will flash 'CONFIG LOCK'. When you complete the change and save the configuration file, the lock will be released and the status bar will be cleared.

The tree pane will display **'[Out Of Date]'** on other servers in the domain. To refresh the status on the out-of-date servers, on the **Service** menu, click **Stop**, and then click **Start**. The status bar will display **'OUT OF DATE'** if SNA Manager is out-of-date. On the **File** menu, click **Refresh**.

Saving Configuration Changes

If the SNA Manager is opened in the domain, it will attempt to obtain an exclusive lock on the configuration file when it is opened (unless opened in read-only mode). It is recommended that you do not run SNA Manager until all servers in the domain have been upgraded to Host Integration Server.

SNA Manager will lock the configuration file when a configuration change is initiated. When the change is completed and the configuration file is saved, then the configuration file lock will be released. The status bar will display **OUT OF DATE** on other servers in the domain. To refresh the status on the out-of-date servers, SNA Manager must be closed and reopened.

Server Configuration Setup Problems

If a server is set up incorrectly, SNA Manager allows the administrator to make changes. Right-click on the server you want to modify.

1. Open SNA Manager, and select a server.

2. On the **Server Properties** page, click on the **Server Configuration** tab.

3. Click **Change**. From **Server Configuration**, you can change the subdomain, role, and transport protocol.

4. Restart the server for the changes to take effect.

Problems with Other Features

In This Section

DLS Status Problems

If you get a message stating that no DLS service is available for a given server, you may not have access to the Registry for that server. For DLS status (or **dlsstat** on the command line) to display the available link services, you need to have access to the Registry on the system that you specify in DLS status.

To correct this problem, see if you have any access to the Registry keys in the HKEY_LOCAL_MACHINE window for a remote server: run **regedt32** on the Host Integration Server computer on which you ran dlsstat and connect to one of the remote computers. If everything in the Registry window is unavailable (dimmed out), dlsstat will not work properly. Make sure you have access to the Registry before running DLS status again.

Virus-Checking Tool Problems

When you are running Host Integration Server with some independent software vendor virus checking tools, SNA link services might fail to start due to the following error:

Event 128: "Can't load IHV DLL xxx.dll"

where xxx.dll is the specific DLL

It is recommended that any "real-time scanning" options be disabled on the virus checker before starting Host Integration Server to prevent local file access slowdowns.

Troubleshooting Network Integration

In This Section

Troubleshooting IP-DLC Link Services

IP-DLC uses the Host Integration Server Trace Utility.

Troubleshooting Transaction Integrator

Use the information in this section to troubleshoot problems in your COM-based or .NET Framework-based client application and Transaction Integrator (TI).

General Tips

Here are some general tips to help you troubleshoot problems with COM-based applications:

- Collect and analyze error message information.

- Collect and analyze Windows Event Log information. Gather all available error information: HRESULT values, error strings, and information in the Windows Logs. For more information, see Checking the Windows Event Logs for Errors.

COM Error Messages

To understand COM application error messages, you need to understand the roles of HRESULT and failfast.

At the end of every method call, the called object sends a four-byte status code (an HRESULT) to its client. For example, most methods return a value of 0x00000000, which signifies that the call completed successfully. However, other numbers can be returned, each of which has a specific meaning. Often these numbers are displayed to the user, in which case they can help you understand the problem and how to correct it.

If a grave error occurs, COM can shut down the process that caused the problem. This sort of behavior is called a *failfast*. It is performed to guard system and data integrity. A general protection fault (GPF) or an Access Violation in a COM+ application causes a failfast. When a failfast occurs, there is no error message; Windows just shuts down the offending process. When a failfast occurs, COM logs a lot of error information to the event log. Thus it is important to search the event log for data if a problem is occurring.

To find out more about specific error messages:

- Search for the error text in this document as well as on the Web. For more information, see Additional Help with Events and Errors.

- Identify the HRESULT values. If you see a long number along with the error message, that number is probably an HRESULT. All HRESULT values are displayed in either hexadecimal or decimal format. In hexadecimal format, an HRESULT is eight hexadecimal digits (0-9, A-F.), such as 0x400FAB23 or 0x80001234. In decimal format. An HRESULT is a ten-digit number, and it will probably be negative, such as -2146889713.

  The thirty-two bits in an HRESULT are numbered from 31 to 0, and bit 31 is the highest bit.

- Once you have the HRESULT, use it to:

  - Confirm that the HRESULT signals failure. If bit 31 is 0, then this is a success or an informational HRESULT. For example, 0x00000000 signals success. If bit 31 is not zero, the HRESULT signals a warning or an error. For example, 0x80000000 is an error HRESULT.

  - Look for the source of the problem. HRESULT values contain a facility code that specifies the error source. The facility code is in bits 16-27 of the HRESULT. In the hexadecimal format, the facility code is in the Z positions in 0x0ZZZ0000. Following are the meanings of the various facility codes: 0 = COM, 1 = RPC, 2 = Dispatch, 3=Storage, 4 = ITF/?, 7 = Win32, 8 = Windows, 9=SSPI, 0xA=ActiveX Controls/MDAC/user code/?, 0xB=Certification, 0xC = Internet, 0xD=Mediaserver, 0xE = MSMQ, 0xF=SetupAPI, 0x10 = Scard, 0x11 = COM+.

In This Section

Cannot Use Save Command in TI Designer

Trouble Creating an Object

Case Discrepancies in Assigned Names

Visual Basic Limitation on Number of Parameters Per Method

How to Resolve Transactions Manually

Mainframe Issues Affecting Transaction Recovery

Allocation Failure

How to Start and Stop DTC or SNA LU 6.2 Resync TP

Newly Deployed Components Not Recognized

Data Type Conversion Errors

Memory Leak When Using User-Defined Types in User-Defined Types

Avoiding Data Translation

Using TI User-Defined Types with the .NET Framework and Visual Studio

How to Use Variable Length Recordsets with Transaction Integrator

See Also

Tasks

Additional Help with Events and Errors

Other Resources

Windows Event Viewer

How to Check the Windows Event Logs for Errors

The Windows Event Log keeps a record of the system's behavior. It contains:

- Informational events that signal normal system function. For instance, certain services log an event whenever they start or shut down.
- Warning events that signal issues that can be problematic but are not actual errors.
- Errors. If you find any error events in your logs, this indicates a problem.

COM+ and .NET often log error events when they detect a problem in an application. This is their main way of signaling a problem or giving diagnostic information. Therefore, understanding a COM+ or .NET problems begins with searching the event log for errors.

To search for errors in the event logs

1. Look for a red circle that contains an x. Errors generally indicate a serious problem, so you should troubleshoot them before moving on to the specific problem.
2. Search the event logs for COM+ errors. COM+ generally logs its events in the Application and System logs. You can determine which errors are from COM+ by looking at the Source column. Errors with a source heading of COM+ or MSDTC are COM+ errors. Explore the COM+ errors by double-clicking the error to bring up an error dialog box that contains all the error information: Source, Machine, Date, Time, and Event ID. At the bottom of the error dialog box is an error description. Read this carefully as it will explain the error and even recommend a remedy. Also, check the Data area on the error dialog box; it may contain additional useful binary information. If you find an HRESULT, analyze it.
3. Determine whether the error contains a call stack. A call stack is a piece of text describing what the application was doing when the error occurred. It begins with a Call stack line. If there is a call stack, determine which .dll file caused the error. Each line in the call stack begins with the name of a .dll file and ends with an exclamation mark or a plus sign.

   The call stack shows a cause and effect chain. The .dll file listed in the top line in the call stack is the direct cause of the error, and .dll files listed on lines below it are indirect causes of the error.

See Also
Other Resources
Windows Event Viewer

Tracing and Debugging

Transaction Integrator (TI) provides low impact performance monitoring for call volumes, processing time, and data throughput capable of isolating each major participant (host, TI, the .NET Framework, and client) to the granularity of transaction type and programming model supported. This support is turned on or off by the administrator online.

See Also

Other Resources

Using the SNA Trace Utility

Client Application Does Not Start but No Error Given

If you double-click the .exe file for your Visual Basic client application and nothing happens but no error message appears, the problem may be that you did not deploy the TI component that your client application is attempting to use.

Normally, if a TI component type library is not registered (that is, it has not been deployed in a COM+ application), you will receive error message number 429 that says, "ActiveX component cannot create object." However, no error appears at all if the unregistered TI component contains a user-defined type (UDT) and that UDT is referenced in the Visual Basic client application.

To resolve this problem, deploy the TI component in a COM+ application to automatically register it. Then your Visual Basic client application will work.

Trouble Defining a Recordset for Web-Based Applications

In TI, a *recordset* consists of tabular data defined in COBOL source code on the mainframe. Tabular data is defined by a group item containing an OCCURS clause in the COBOL data area. When you import a COBOL data area into TI Designer, the following COBOL-to-Automation conversions take place:

- The COBOL data area defines the parameters of the newly created method and the members of any recordsets.
- The group item that defines the table (contains the OCCURS clause) is represented as both the type definition of the method's recordset and a method parameter.
- Other group items are represented as method parameters.
- Elemental data items (definitions of the table fields) are represented as the recordset's members.

The following COBOL data area describes the type library for a Web-based application that uses a CICS LINK remote environment. The application returns information on up to six accounts for each customer name and matching PIN entered as input.

```
01          DFHCOMMAREA.
*                     ACCTINFO IS (INPUT, OUTPUT)
            05        ACCTINFO OCCURS 6 TIMES.
                      10 ACCOUNTNUMBER                    PIC X(6).
                      10 ACCOUNTTYPE                      PIC X(20).
                      10 CURRENTBALANCE                   PIC S9(13)V9(2) COMP-3.

                      10 INTERESTBEARING                  PIC S9(4) COMP.
                      10 INTERESTRATE                     COMP-1.
                      10 MONTHLYSVCCHG                    PIC S9(13)V9(2) COMP-3.

*                     NAME IS (INPUT, OUTPUT)
                      05                                 NAMEPIC X(30).
*                     PIN IS (INPUT, OUTPUT)
                      05                                 PIN PIC X(10).
```

When imported into TI Designer, the data area's group items are treated as the parameters of the newly created method. However, because of Remote Data Service (RDS) requirements for Web-based applications, the group item that defines the table must be defined as the method's return value, not as a method parameter. To define the method correctly, you must manually redefine this group item (ACCTINFO in the previous example) as a return value.

Before you import the COBOL data area, note the number of rows specified in the OCCURS clause. After you have imported the COBOL data area, use the following procedure to define a recordset for Web-based applications.

To define a recordset for a Web-based application

1. Start TI Designer.
2. In the console tree, double-click the **Recordsets** folder to verify that TI Designer created the type definition of the recordset. The type definition's name is taken from the group item that defined the table in the COBOL source code.
3. Double-click the **Methods** folder, and click the method's name. Verify that the recordset parameter is displayed in the details pane. The parameter name should match the name of the recordset's type definition.
4. On the **Edit** menu, click **Unlock** to unlock the method.
5. In the details pane, delete the recordset parameter.
6. Right-click the method, click **Properties**, and then click the **Automation Definition** tab.
7. Click the name of the recordset's type definition in the **Return Type** box.

8. Click the **Recordsets** tab.
9. In the **Group-Item Maximum** box, type the number of rows specified in the COBOL source code, and then click **OK**.

For detailed information about recordsets, see the ActiveX Data Objects (ADO) and Remote Data Service (RDS) documentation included when you installed Microsoft Data Access Components (MDAC).

Cannot Use Save Command in TI Designer

Before updating a Transaction Integrator (TI) component, consider how your changes affect applications that call the component's methods. If, for example, you add a parameter to an existing method or change the direction of a parameter, current applications will no longer be able to call the method successfully. To prevent unintended overwrites of TI components, the **Save** command in TI Designer is disabled for any component that has been added to a COM+ Application.

To save modified components that no longer need to work with existing applications, first display the properties of the component's interface and change the component's ProgID (*programmatic ID* in Visual Basic terminology). Then use the **Save As** command to save the component under a different file name. The new component can be used with later applications, and the original component can be used with existing ones.

You can modify a component in such a way that it will continue to work with existing applications. (For example, you can add a method to the component's interface without breaking existing applications.) In this case, how you save the modified component depends on whether existing applications connect to the component through declarative binding or late binding. Declarative-bound applications connect to a component when the application is compiled. Declarative-bound applications recognize a required component by the component's class identifier (CLSID). Late-bound applications connect to a component at run time. Late-bound applications recognize a required component by the component's ProgID.

If you modify a component accessed by a late-bound application, you can save your modifications with the Save As command. As long as you do not change the component's ProgID, existing late-bound applications can still access the component. (It is recommended that you change the component's minor version number on the properties page for the component's interface. The minor version number is the number to the right of the decimal point in the Version box on the properties page.)

Because the Save As command changes a component's CLSID, you must use the Save command to save modifications to any component that will continue to support existing declarative-bound applications. This means that before you open the component's .tlb file in TI Designer, you must delete the component from its COM+ application. Use the following procedure to modify a TI component currently deployed in a COM+ application.

To modify a TI component currently deployed in a COM+ application

1. Before opening the component in TI Designer, start TI Manager.
2. In the **Component Services** folder of TI Manager, shut down the server process on the computer where the TI component's COM+ application resides.
3. Delete the TI component from its COM+ application.
4. In TI Designer, open the TI component's .tlb file.
5. Modify the TI component, and then change the component's minor version number on the properties page for the component's interface.
6. Save the component. Because you deleted the component from its COM+ application before opening it in TI Designer, the **Save** command is enabled.
7. Deploy the new version of the TI component in a COM+ application.

Note

If you modify the TI component before deleting it from its COM+ application, your only recourse is to close the component's .tlb file without saving it, and then follow the procedure.

Note

If you are not sure whether the component is used by late-bound or declarative-bound applications, you can protect existing applications by using the above procedure instead of using the **Save As** command.

Trouble Creating an Object

When running your application, if you encounter the message "ActiveX component cannot create object," try the following to correct the problem:

- Verify that the TI component library is deployed in a COM+ application.
- Verify that the application references the correct ProgID.
- Verify that ActiveX Data Objects (ADO) is installed. This is necessary if the object you want to create is a recordset object. ADO is a subcomponent of Microsoft Data Access Components (MDAC), which is an integral part of Windows 2000 or Windows Server 2003.

Case Discrepancies in Assigned Names

Use care when assigning names to any part of a type library in TI Designer. You can assign case-sensitive variants of the same name. For example, you can assign a name of *myparam* to a parameter of one method, and *MyParam* to a parameter of another method. However, when you save the type library, the name of the second parameter will be changed to match the case of the first. In the previous example, both parameters would be saved as *myparam*. Such unexpected changes in case are potentially confusing, although they will not affect the operation of your application.

Visual Basic Limitation on Number of Parameters Per Method

If you attempt to compile a Visual Basic 6.0 application containing a component with more than 60 parameters per method, you can encounter the following error: "Subscript out of range." This error occurs with applications that connect to the component through declarative binding. You can avoid this error by switching to late binding. If, for the sake of performance, you want to maintain declarative binding, you can modify the component to aggregate its parameters within a single recordset. The recordset will then be treated as a single parameter and will not cause the error the next time you compile.

How to Resolve Transactions Manually

The following procedures describe how to resolve a transaction manually when it cannot be committed or aborted by the system due to a resynchronization failure following restoration of services between the Windows 2000 or Windows Server 2003 and IBM LU 6.2 systems. Such resynchronization failures can occur, for example, if CICS makes a heuristic decision to commit or abort a transaction. CICS versions prior to 5 will do this. Typically, TI and Microsoft Distributed Transaction Coordinator (DTC) will automatically resolve all in-doubt transactions when service between the systems is restored. However, if resynchronization and recovery cannot be automatically achieved for any reason, you can resolve transactions manually by using one of the following procedures.

To resolve a transaction manually

1. For transactions in the **Only Failed Remain to Notify** state or in the **Cannot Notify Committed** state:

   The Only Failed Remain to Notify and the Cannot Notify Committed states indicate that the transaction has committed, but some subordinate Microsoft DTC or IBM LU 6.2 systems have not been notified.

   a. Start TI Manager, and navigate to **Transaction List** in the console tree's **Component Services** folder in Windows 2000 or Windows Server 2003.

   b. In the **Transaction List** details pane, right-click the transaction that is in the Only Failed Remain to Notify or Cannot Notify Committed state.

      This will display the parent DTC and the subordinate DTC and IBM LU 6.2 systems for the transaction.

   c. Force the transaction to commit on each subordinate system.

   d. Return to the DTC that shows the Only Failed Remain to Notify or Cannot Notify Committed state, and force that DTC to forget the transaction.

Caution
Do not manually forget a transaction until all subordinate systems have been notified of the transaction outcome.

2. For transactions in the **Aborted** state or in the **Cannot Notify Aborted** state:

   The Aborted and Cannot Notify Aborted states indicate that the transaction has aborted. If a transaction remains in one of these states for an extended period of time, this indicates that some subordinate DTC or IBM LU 6.2 systems have not been notified of the transaction's outcome.

   a. Start TI Manager, and navigate to **Transaction List** in the console tree's **Component Services** folder in Windows 2000 or Windows Server 2003.

   b. In the **Transaction List** details pane, right-click the transaction that is in the Aborted or Cannot Notify Aborted state. This will display the parent DTC and the subordinate DTC and IBM LU 6.2 systems for the transaction.

   c. Force the transaction to commit on each subordinate system.

   d. Return to the DTC that shows the **Aborted** or **Cannot Notify Aborted** state, and force that DTC to forget the transaction.

Caution
Do not manually forget a transaction until all subordinate systems have been notified of the transaction outcome.

For more information about resolving transactions manually, see the Windows 2000 or Windows Server 2003 documentation.

Note

Resolving a transaction manually does not apply to TCP/IP because the IBM TCP/IP protocol does not currently support ACID (atomic, consistent, isolated, durable) transactions.

Mainframe Issues Affecting Transaction Recovery

In some situations, TI cannot process new transactions with a remote environment. This can be correct behavior. For example, if TI exception 1227 is returned to a client application or logged in an event, and the HRESULT is 8004D110, it indicates that new transactions with this remote environment cannot be accepted because previous transactions were not resolved after a communications failure.

When the two-phase commit process does not complete, CICS must hold the transaction in the In-Doubt state until communications are re-established. Then TI will perform recovery protocols to ensure that the transaction is in the same state at all nodes. CICS must be configured correctly for this to occur.

If CICS terminates unexpectedly, and then is restarted in a cold state, there is no memory in its log of any transactions that have not completed. Therefore, these transactions cannot be automatically recovered to a consistent state. Verify that all transactions have completed before stopping CICS, or configure CICS for a warm restart using the same log so that any pending transactions can be recovered.

CICS Transaction Server allows the administrator to specify a Wait Time in the In-Doubt Attributes of a transaction. Be sure to specify a value that is adequate to allow communications to be re-established in most cases. If this timeout elapses before all transactions left in the In-Doubt state have been recovered, CICS will make a heuristic decision to resolve them locally. If this decision conflicts with the decision made for the transactions by Microsoft DTC (Distributed Transaction Coordinator), new transactions cannot be started until the outcome of the previous transactions have been manually overridden.

In CICS versions prior to CICS Transaction Server, there is no Wait Time in the Recovery attributes. Assigning the Wait value to the In-Doubt attribute does not cause CICS to place the transaction in the state requested by TI when recovery is attempted. If you are using these versions of CICS, set the In-Doubt attribute to Backout or Commit. If a resulting heuristic decision is incorrect and prevents new transactions from beginning, override the outcome of the transaction using DTC.

Examine the Windows Event Log for messages from the SNA LU 6.2 Resync TP service indicating that transactions were not recovered successfully. Follow the suggested actions. Use Microsoft Transaction Server's Transaction List window to show pending transactions. Right-click the transaction to show its properties. Resolve it to agree with the state that CICS was configured to select heuristically, or to the backed-out or aborted state if CICS terminated unexpectedly and started up cold. The event in the log identifies the transaction and the state chosen by CICS.

Note
This does not apply to TCP/IP because TCP/IP does not support ACID (atomic, consistent, isolated, and durable) transactions.

See Also
Tasks
How to Resolve Transactions Manually

Allocation Failure

If you used one of the modes that come pre-packaged with Host Integration Server, that mode can attempt to dynamically create a session with the partner. This fails with an abnormal CNOS reply, "mode name not recognized." The error returned by the APPC API is ALLOCATION FAILURE - RETRY. This leads to an error stating: "The TI LU 6.2 transport failed to allocate conversation, make sure that host LU (hostname) is active, and retry the operation."

You will see this same error when you try to create a session with a CICS region that is not active.

To avoid this problem, upon entering SNA Manager in Host Integration Server, add the LUs and modes that you will be using and delete any pre-supplied modes that you will not be using.

How to Start and Stop DTC or SNA LU 6.2 Resync TP

If necessary, you can manually stop and start the Microsoft Distributed Transaction Coordinator (DTC) and/or the SNA LU 6.2 Resync TP services by clicking **Services** in the **Administrative Tools** menu in Windows 2000 or Windows Server 2003.

Use the following procedures to start or stop services in Windows 2000 or Windows Server 2003. It is important to start and stop the services in the precise order given.

To stop DTC or SNA LU 6.2 Resync TP in Windows 2000 or Windows Server 2003

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Services**.
2. Right-click **SNA LU 6.2 Resync TP**, and then click **Stop**. If prompted, click **Yes**.
3. Right-click **SnaBase**, and then click **Stop**. If prompted, click **Yes**.
4. Right-click **DTC**, and then click **Stop**. If prompted, click **Yes**.

To start DTC or SNA LU 6.2 Resync TP in Windows 2000 or Windows Server 2003

1. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Services**.
2. Right-click **DTC**, and then click **Start**.
3. Right-click **SnaBase**, and then click **Start**.
4. Right-click **SNA LU 6.2 Resync TP**, and then click **Start**.

Newly Deployed Components Not Recognized

You can deploy new components in a COM+ application while other components within that application are servicing clients. However, the Component Services run-time environment will not recognize and service these newly deployed components until the Component Services server process associated with the application is stopped. As a result, the following error is returned: "An attempt was made to launch a server process for a package that was already actively supported by another server process on this computer."

To avoid this problem, run the **Shutdown Server Processes** command after each test run and before adding a new component to a COM+ application to do another test run. This stops the active process for the package.

⚠Warning
The **Shutdown Server Processes** command interrupts all components in all COM+ applications for a given machine. Theref ore, you should carefully consider the consequences of this action.

Data Type Conversion Errors

A message indicating that a data type conversion could not take place uses a numeric code to identify the data type. The following tables translate the numeric codes into their equivalent variant data types (for Visual C++) and Automation data types (for Visual Basic).

Numeric Code	Variant Data Type	Automation Data Type
0x0000	VT_EMPTY	nothing
0x0002	VT_I2	2-byte signed int
0x0003	VT_I4	4-byte signed int
0x0004	VT_R4	4-byte real
0x0005	VT_R8	8-byte real
0x0006	VT_CY	currency
0x0007	VT_DATE	date
0x0008	VT_BSTR	OLE Automation string
0x0009	VT_DISPATCH	IDispatch * (currently only for recordset pointer)
0x000b	VT_BOOL	True=-1, False=0
0x000c	VT_VARIANT	VARIANT *
0x000e	VT_DECIMAL	16-byte fixed point
0x0011	VT_UI1	unsigned char
0x0018	VT_VOID	C style void
0x001b	VT_SAFEARRAY	(use VT_ARRAY in VARIANT)
0x001d	VT_USERDEFINED	user-defined type

Arrays of the following types have these codes:

Numeric Code	Variant Data Type	Automation Data Type
0x2000	VT_EMPTY	nothing
0x2002	VT_I2	2-byte signed int
0x2003	VT_I4	4-byte signed int
0x2004	VT_R4	4-byte real
0x2005	VT_R8	8-byte real
0x2006	VT_CY	currency

0x2007	VT_DATE	date
0x2008	VT_BSTR	OLE Automation string
0x2009	VT_DISPATCH	IDispatch * (currently only for recordset pointer)
0x200b	VT_BOOL	True=-1, False=0
0x200c	VT_VARIANT	VARIANT *
0x200e	VT_DECIMAL	16-byte fixed point
0x2011	VT_UI1	unsigned char
0x2018	VT_VOID	C style void
0x201b	VT_SAFEARRAY	(use VT_ARRAY in VARIANT)
0x201d	VT_USERDEFINED	user-defined type

The following types are passed by reference:

Numeric Code	**Variant Data Type**	**Automation Data Type**
0x4000	VT_EMPTY	nothing
0x4002	VT_I2	2-byte signed int
0x4003	VT_I4	4-byte signed int
0x4004	VT_R4	4-byte real
0x4005	VT_R8	8-byte real
0x4006	VT_CY	currency
0x4007	VT_DATE	date
0x4008	VT_BSTR	OLE Automation string
0x4009	VT_DISPATCH	IDispatch * (currently only for recordset pointer)
0x400b	VT_BOOL	True=-1, False=0
0x400c	VT_VARIANT	VARIANT *
0x400e	VT_DECIMAL	16-byte fixed point
0x4011	VT_UI1	unsigned char
0x4018	VT_VOID	C style void
0x401b	VT_SAFEARRAY	(use VT_ARRAY in VARIANT)

0x401d	VT_USERDEFINED	user-defined type

Arrays of the following types are passed by reference:

Numeric Code	Variant Data Type	Automation Data Type
0x6000	VT_EMPTY	nothing
0x6002	VT_I2	2-byte signed int
0x6003	VT_I4	4-byte signed int
0x6004	VT_R4	4-byte real
0x6005	VT_R8	8-byte real
0x6006	VT_CY	currency
0x6007	VT_DATE	date
0x6008	VT_BSTR	OLE Automation string
0x6009	VT_DISPATCH	IDispatch * (currently only for recordset pointer)
0x600b	VT_BOOL	True=-1, False=0
0x600c	VT_VARIANT	VARIANT *
0x600e	VT_DECIMAL	16-byte fixed point
0x6011	VT_UI1	unsigned char
0x6018	VT_VOID	C style void
0x601b	VT_SAFEARRAY	(use VT_ARRAY in VARIANT)
0x601d	VT_USERDEFINED	user-defined type

Memory Leak When Using User-Defined Types in User-Defined Types

If Microsoft COM Transaction Integrator (COMTI) is configured to use a "Customer Information Control System (CICS) or Information Management System (IMS) by using TCP/IP" Remote Environment (RE), and a client application repeatedly calls the COM+ component, which in turn instantiates COMTI objects by using user-defined types, eventually the application might fail and return the following error message:

Method %1 of Object %2 failed

Note
Other REs may exhibit the same problem.

If you use Microsoft Windows System Monitor to log data for the Private Bytes and Working Set of the Process object, a memory leak occurs.

The problem is caused by having Occurs Depending On (ODO) arrays in a user-defined type. Specifically, a call is made to obtain a VarDesc structure from a type library, and a free method call is never issued to release the memory back to the operating system.

Avoiding Data Translation

In some circumstances, you may want the Transaction Integrator runtime to pass untranslated data to or from the mainframe. To do this, set up an array of PIC X Untranslated bytes.

TI supports many data types, however, you may not always want TI to translate or interpret the data.

To configure a byte array of PIC X Untranslated bytes, follow these steps:

1. Open the COMTI Component Builder.
2. Unlock the COMTI component.
3. Select the properties for the parameter that you want to change.
4. On the Automation tab, set the data type to Byte.
5. On the COBOL Definition tab, set the COBOL Definition to PIC X Untranslated.
6. On the Arrays tab, set the array to be a Single Dimension Array, and set the maximum size of the array equal to the expected number of bytes.
7. Lock the component.

After the last step is complete, TI will pass the bytes in the array to the calling program as untranslated binary data.

Because MTI passes the bytes as untranslated binary data, the interface program must take into account the newly modified parameter. You can use this procedure if, for example, the characters coming from or going to the host are outside the range of the translation table. By following the steps earlier in this section, you can implement a custom translation table in code that handles the data.

If a variable-sized array is to be transferred, follow these steps:

1. Set the array size to the maximum number of characters ever to be exchanged.
2. On the Advanced tab of the method properties, set the Data buffer options as follows:
    a. Final field from host is Bounded.
    b. Final field to host is Bounded.

Using TI User-Defined Types with the .NET Framework and Visual Studio

TI supports the use of user-defined types for parameters and return values. A few things to note about user-defined types:

- They are defined by using the TI Designer application.
- They work very well in a Visual Basic 6 development environment.
- They also work well in a Visual C++ 6 development environment.

However, by default, user-defined types do not work in an application that is based on the .NET Framework, version 1.0. For example, if a UDT is passed to TI by a .NET Framework application, the following error will be logged by TI:

(102) COM Transaction Integrator reported the following exception to the client:

Component: TestUDT.TestUDT.1

Method: Method1

Exception description:

(1205) COM Transaction Integrator detected an error on parameter 1 prior to the remote operation. COM Transaction Integrator could not convert from variant type (0x4009) to (0x401d) (0x80020008). Contact the application programmer. Install the correct component library for this component in Microsoft Transaction Server.

This call does not work because the COM Interop layer of the .NET Framework 1.0 marshals the user-defined type as a VT_DISPATCH type instead of as a VT_RECORD type or VT_USERDEFINED type.

To successfully use COMTI user-defined types with a .NET Framework 1.0 application, that application must implement a custom marshaller. This means that the application must build the VT_RECORD structure itself before handing it off to COMTI. This type of coding requires advanced knowledge of COM and .NET Framework data types and marshaling between those types.

In the .NET Framework (version 1.1), the COM Interop layer automatically marshals user-defined types as VT_RECORD types. This means that .NET Framework 1.1 applications can use COMTI and user-defined types without implementing any custom marshaling.

Visual Studio .NET 2003 uses this version of the .NET Framework. Therefore, applications that are written in Visual Studio .NET 2003 can also use COMTI and user-defined types without any extra coding.

To use TI from a .NET Framework 1.1 application:

1. Create a .NET interop assembly for the COMTI type library. To do this, you can use either the TLB Importer Utility (Tlbimp.exe) or the Add Reference dialog box in Visual Studio.
2. Register the newly created .NET Framework interop assembly by using the Register Assembly Utility (Regasm.exe). If the assembly is not registered, the calling application can pass user-defined types to COMTI but will receive an exception when COMTI tries to pass a user-defined type back to the calling application.
3. Add a reference to the .NET interop assembly in the .NET Framework 1.1 application.

How to Use Variable Length Recordsets with Transaction Integrator

When data of variable length with field type of recordset is sent to the host by way of COM Transaction Integrator (COMTI), COMTI sends the final field to the host padded with NULL characters (0x00) up to the defined maximum size of the field.

When the last parameter in the type library is a recordset that refers to an OCCURS DEPENDING ON value, the null values will be passed to the mainframe unless the option Final field from host is variably sized (Return Value) is selected.

The Final field from host is variably sized (Return Value) option is located in the method's Properties on the Advanced tab.

The following trace snippet from an SNA Server Logical Unit (LU) 6.2 message trace shows the problem when the Final field from host is variably sized (Return Value) option is not selected:

PVI ---------- 14:08:18.0343

PVI 0A1F0001->0102FEB1 LU 6.2

PVI MSGID:SWAT MSGTYP:NTEOD Sense1:2E00

PVI Sense2:0100

PVI

PVI ---- Header at address 01196054, 14 elements ----

PVI 0B012E00 01000400 000E0000 01007E01 <.....~.>

PVI

PVI ---- Element at address 01B89574, start 13, end 268 ----

PVI 2B0502FF 0003D100 0004C3E2 D4C9001A <+.....J...CSMI..>

PVI 11C5C3C9 C2D4C8C8 F14BE3F2 F2F1F5F8 <.ECIBMHH1KT22158>

PVI C4C407D0 0B100E08 00010090 0012F20D <DD.....2.>

PVI 85020201 02000004 C3E2D4C9 12430E02 <e.....CSMI.C..>

PVI 000006E4 00070000 0104D4E2 E3E7000B <...U.....MSTX..>

PVI 02C3D3D3 D3C9E2C3 D3000504 14AE14B1 <.CLLLISCL.....>

PVI 06D4C5D5 C1404040 40404040 40404040 <.MENA@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI 40404040 40404040 40404040 40404040 <@@@@@@@@@@@@@@@@>

PVI

PVI ---- Element at address 01B89B28, start 13, end 268 ----

PVI 40000000 00000000 00000000 00000000 <@.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 0000F0F0 F0000000 00000000 00000000 <..000.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI

PVI ---- Element at address 01B88E9C, start 13, end 268 ----

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI

PVI ---- Element at address 01B815B4, start 13, end 268 ----

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

PVI 00000000 00000000 00000000 00000000 <.....>

Development

This software development kit (SDK) section of Microsoft Host Integration Server Help provides information for the programmer writing applications for Host Integration Server 2009.

Using Help in a Developer Environment

Host Integration Server Help contains features that you can use to display the developer documentation in your preferred language and to link between Host Integration Server Help and Visual Studio Help.

Using Language Filtering

The Host Integration Server .NET Framework Class Reference and COM Object Reference provide signatures and code examples in multiple programming languages.

You can customize your view of the content in the reference pages to display information only in your preferred programming language. To select a custom language, use the Language Filter button in the upper-left corner of the reference page. After it is enabled, the language filtering is persisted until it is changed. When language filtering is enabled on a page, the name of the language follows the title in the blue bar at the top of the page.

Viewing Host Integration Server Help in Visual Studio

You can view Host Integration Server Help in three ways:

- In Visual Studio, select **Contents** on the **Help** menu.
- In the Visual Studio Help, select **Microsoft Visual Studio** from the **Programs** menu.
- In the stand-alone Microsoft Host Integration Server Help, select **Host Integration Server Documentation** from the **Programs, Microsoft Host Integration Server** menu.

If you are developing in Visual Studio, either of the first two methods are recommended. Using either of these methods enables integration between Host Integration Server Help and Visual Studio Help; this integration is extremely useful when navigating class relationships across the documentation sets. If you view Host Integration Server Help outside of Visual Studio, read the following tip to properly render links to Visual Studio.

Linking between Host Integration Server Help and Microsoft Visual Studio Help

When a member is inherited from the .NET Framework Base Class Library, two links are provided, as shown in the following example:

Finalize (inherited from **System.Object**)	For additional information about the System namespace, see .NET Framework Help available from Visual Studio or go to http://go.microsoft.com/fwlink/?LinkID=9677.

If you are viewing Host Integration Server Help in Visual Studio or in the Visual Studio Help, the link in the left column goes to the exact member page. If you are viewing Host Integration Server Help outside of Visual Studio, use the link in the right column to go to the System namespace page in MSDN Library. Note that the link will not go to the specific member page.

In This Section

Programmer's Guide

Host Integration Server 2009 provides comprehensive bidirectional services for integrating Microsoft Windows with legacy systems.

Most of the services provided by Host Integration Server expose a programming interface, which enables you to extend the functionality of the product and integrate it more tightly in your own environment. This guide describes these interfaces and provides guidance on how to use them.

In This Section

Application Integration Programmer's Guide

This section of the Microsoft Host Integration Server 2009 Software Development Kit (SDK) provides information required to develop software to integrate COM and .NET applications with Customer Information Control System (CICS) and Information Management System (IMS) transactions on IBM mainframe and AS/400 computers.

Transaction Integrator (TI) enables developers to integrate mainframe-based transaction programs (TPs) with component-based Microsoft Windows applications. With Transaction Integrator, you can integrate existing mainframe-based TPs with Windows-based COM or distributed COM (DCOM) applications. You may not have to modify your mainframe TP if the business logic is separate from the presentation logic. The wizards available in the TI Designer and TI Manager guide you through the process, step-by-step.

Transaction Integrator is appropriate when you need a synchronous or transactional solution where both systems being integrated are running at all times. For applications only requiring an asynchronous integration solution, a messaging-based solution using the MSMQ-MQSeries Bridge is preferred over Transaction Integrator.

Applications that integrate message queuing and that use MSMQ-MQSeries Bridge in a Host Integration Server 2009 environment can be developed using several different development tools and application programming interfaces including the following:

- C or C++ applications that use the MSMQ-MQSeries Bridge Extensions to extend the MSMQ-MQSeries Bridge.
- Microsoft Visual Basic applications that use MSMQ-MQSeries Bridge Extensions to extend the MSMQ-MQSeries Bridge.

To use this guide effectively, you should be familiar with the following:

- Microsoft Host Integration Server 2009
- Microsoft Windows 2000 or later
- Message Queuing
- IBM MQSeries

Depending on the application programming interface and development tools used, you should be familiar with the following:

- Microsoft COM objects
- ASP
- ASP.NET

For API references and other technical information for the Transaction Integrator, see the Programmer's Reference section of the SDK.

For sample code using the Transaction Integrator, see Application Integration Samples in the Samples section of the SDK.

For information about how to tune your system to get the best possible performance from Transaction Integration, see the Transaction Integrator Performance Guide in the Operations section.

In This Section

This section contains the following topics

- Application Integration Development Tools
- Application Integration Programming

Application Integration Development Tools

This section contains the following topics:

In This Section

- How to Install Host Integration Server Designer
- How To Migrate from Earlier Versions of Host Integration Server
- How To Report Errors in Host Integration Server Designer

How to Install Host Integration Server Designer

Transaction Integrator (TI) Designer is installed and configured by the Host Integration Server 2009 Installation Wizard. You can install and configure TI Designer at the time you first install Host Integration Server 2009, or you can use the Host Integration Server 2009 Installation Wizard at a later time to add TI Designer. The Host Integration Server 2009 Installation Wizard installs everything you need, including program files, Help files, sample applications, and other tools. The TI Project template is automatically installed on your computer at <drive>:/Program Files/Microsoft Host Integration Server/System/Projects/.

Note
TI Designer is hosted in the Visual Studio development environment and must be registered in Visual Studio at the time of in stallation. Be sure that Visual Studio is installed on your computer before you install TI Designer.

To install Transaction Integrator Designer

1. Start Host Integration Server Setup and accept all defaults until you reach the **Custom Installation** page.
2. On the **Custom Installation** page, expand the **Application Integration** node.
3. Select **Transaction Integrator Designer** and any other options you want to install.
4. Click **Next**, and then follow the on-screen directions.

See Also

Concepts

TI Designer

How To Migrate from Earlier Versions of Host Integration Server

If you used the COM Transaction Integrator (COMTI) feature of Microsoft Host Integration Server 2000 (or SNA Server 4.0) to integrate Windows-based applications with the mainframe, you have a choice of four options for migrating to Transaction Integrator (TI):

1. Do nothing. Your current type libraries and remote environments (REs) work as they are. Host Integration Server 2009 is fully backward compatible with components created with Host Integration Server 2000 (or SNA Server 4.0).

2. Upgrade all your type libraries at once by dragging them into Microsoft Visual Studio and then clicking **Save All**. Create new REs that have the same characteristics as the old REs using the TI Manager. Associate the updated type libraries with the new REs. This is the easiest upgrade path.

3. Upgrade your type libraries one-at-a-time by opening them in Visual Studio and then clicking **Save As**. Create new REs that have the same characteristics as the old REs using the TI Manager. Associate the updated type libraries with the new REs.

4. Upgrade your type libraries one-at-a-time by creating new type libraries, and then importing the property settings from the old type library. Use TI Manager to create new REs that have the same characteristics as the old REs. Associate the updated type libraries with the new REs. This is the most time-consuming option because each library is created individually.

When a type library is fully upgraded to Host Integration Server 2009, three aspects of the old type library may be updated depending on the upgrade process you used. These are:

- Remote environment class
- Library, interface, coclass, and UDT GUIDs
- Default conversion information defined at the library level. The defaults for converting data types are different in TI than in COMTI.

Depending upon your circumstances and the requirements of your applications, you might want to update one, two, or all three aspects.

User Action	**TI Actions**	**Results**	**When to use**
None. Use existing Type Libraries and REs.	No changes are made to: • RE • Type library GUID • Type library default conversions	• Type library and RE is fully functional in both Host Integration Server 2009 and Host Integration Server 2000. • The properties within the type library and RE can no longer be modified. • New Type libraries created with TI Project cannot be associated with older REs. • Existing COM client works unmodified.	• Initial upgrade to Host Integration Server 2009. • Old type libraries and old REs work and do not need to be modified.

Open and **Save**	• Updates the Remote Environment class.
	
	No changes are made to:
	  • Typelibrary GUIDs
	  • Default conversions
		• The method property **Include Context Parameter** is automatically set to **True**. Client Applications do not need to support the Client Context and work unchanged as Client Context is an optional parameter in the COM environment. In the Visual Studio TI Project, set **Include Context Parameter** to **False** on the method.
		
		• Old type libraries are updated the Host Integration Server 2009. The updated type library can only be associated with an RE created with TI Manager.
		
		• Existing COM client works unmodified.
			• When a new RE needs to be created in the Host Integration Server 2009 environment.
			
			• Using new Host Integration Server 2009 RE features.
			
			• Using new Client context features (Client Security, Persistent Connections, and so forth).
			
			• Moving to the Host Integration Server 2009 supported environment.
			
			• Programming model stays the same.

Open and **Save As**	• Updates the Remote Environment class. • Generates new type library GUIDS. Save does not. • No changes made to the type library default conversions	The method property **Include Context Parameter** is automatically set to **True**. Client Applications do not need to support the Client Context and work unchanged as Client Context is an optional parameter in the COM environment. In the Visual Studio TI Project, set **Include Context Parameter** to **False** on the method. If you migrate an old type library to a .NET assembly, TI also migrates all Visual Basic version 6.0 Automation data types to Visual Basic .NET data types, including: • Integer to Short • Long to Integer • Currency to Decimal • Recordset to datatable • UDT to Structure Client Applications using the **NewRecordSet** function to create disconnected recordsets must modify the code. The **NewRecordset** function is not supported in .NET. Existing COM Clients that use declarative binding need to be recompiled because of GUID changes.	• Creating a new instance of the application for side by side operation with Host Integration Server 2009 with the intent of extending the application. • Programming model stays the same.
Import	• Updates the Remote Environment class • Updates the GUIDs • Adds Host Integration Server 2009 defaults to the new library	• Same results as **Open** and **Save As** • All methods use new library default defined for Host Integration Server 2009 • This is the only way to migrate type libraries to Assemblies.	• Same as **Save As** • Programming model change • Target environment change (CICS, IMS, AS/400) • Initiation change (WIP, HIP) • Platform change (COM, .NET)

See Also
Concepts
TI Designer

How To Report Errors in Host Integration Server Designer

The standard error-reporting mechanism in Visual Studio is through the **Output** window. This window has been extended to allow for error reporting. For example, if an operation could not be completed, an error message is logged in the **Output** window; the user can then double-click the error message to see its source in the Host Integration Server Designer.

See Also
Concepts
TI Designer

Application Integration Programming

This section contains the following topics:

In This Section

- Creating an Application using Host Integration Server Designer
- Programming Windows-Initiated Processing
- Programming Host-Initiated Processing
- Application Integration Security Guide

Creating an Application using Host Integration Server Designer

Using the tools for Host Integration Server (HIS) Designer in Visual Studio, you can create an application that uses Transaction Integrator (TI) to communicate with a remote mainframe.

1. Create a new project for your application.

2. Add a library to your project that uses Transaction Integrator.

3. If available, import a Host File.

   A host file is a file that describes the interfaces your application will be programming towards on the remote server. Using HIS Designer, you can create a .dll that describes these interfaces.

4. If necessary, use HIS Designer to make any changes or additions to the interfaces.

5. Deploy the interface.

   Deploying the interface allows you to write code against the interfaces you create in steps three and four.

6. Write your application.

   Your application is simply a standard application that includes a reference to the deployed .dll.

7. Test and modify your code.

   If necessary, you many need to undeploy the assembly in order to update the interfaces.

One you are finished testing your application, you can move your application to a staging or production server. If you want to use the BizTalk Adapter for Host Applications, you can add your assemblies to a BizTalk Server export package.

In This Section

How to Create a New Host Integration Server Designer Project

How to Add a Library to a Transaction Integrator Project

How to Import a Host File into a Transaction Integrator Project

How to Modify and Update a Transaction Integrator Interface

How to Deploy a Host File Interface

How to Code a Transaction Integrator Application

How to Test and Modify a Transaction Integrator Application

See Also

Concepts

Host Integration Server Designer UI

How to Create a New Host Integration Server Designer Project

You can create a new Transaction Integrator (TI) Designer project in the Visual Studio 2005 development environment.

To create an HIS Designer Project

1. Click **Start**, point to **Programs**, and then click **Microsoft Visual Studio**.
2. On the Visual Studio **File** menu, click **New**, and then click **Project**.
3. Under **Project Types**, select **Host Integration Projects**.
4. Under **Templates**, select **Transaction Integrator Project**.
5. After **Name**, type the name of the project.

   The name can be a maximum of 256 Unicode characters.
6. After **Location**, type or browse to the location to store the project.

See Also
Concepts
Host Integration Server Designer UI

How to Add a Library to a Transaction Integrator Project

Once you have created the Transaction Integrator (TI) project, you need to add an assembly to the project. Once you have added the assembly, you can import a host file definition.

To add a library to a TI project

1. Click **Project**, and then click **Add .NET Client Library**.
2. On the Add New Item dialog, in the Templates pane, confirm that .NET Client Library is highlighted.
3. In the **Name:** field, type the name of the assembly, and then click **Add**.
4. On the Welcome to the .NET Client Library Wizard page, click **Next**.
5. On the Remote Environment page, select the information that describes the remote environment your application will interact with, and then click **Next**.

   Visual Studio will use this information to optimize your application for the specified remote environment. In contrast, the information you entered in Transaction Manager will be used by Host Integration Server when making a connection.
6. On the Completing the .NET Client Library Wizard page, confirm that the displayed settings are correct, and then click **Create**.

See Also

Other Resources

Creating an Application using Host Integration Server Designer

How to Import a Host File into a Transaction Integrator Project

Once you have created a library in your Transaction Integrator (TI) project, you can import a Host Definition file. After you import the host definition file, you can modify the interfaces using Host Integration Server (HIS) Designer.

To import a Host File into a TI project

1. If you have a Host Definition file (.hcd) file available, you can use the Import COBOL Wizard or the Import RPG Wizard to define your interfaces.

   For more information, see How to Import COBOL into a TI Component or Importing RPG.

2. If you have a previous .NET client library that you want to base your new object on, you can use the Import Library tool to import the library into your project.

   For more information, see How to Import a TI Component.

How to Modify and Update a Transaction Integrator Interface

Once you have imported a host definition file, you can modify and update the interface. Optionally, if you did not start with a host definition file, you can create a new TI interface using the available tools.

Once you have modified the interface, you can deploy and write code against the interface.

To create, modify, or update a TI interface

1. Use HIS Designer to create, modify, or update the TI interface. For more information, see Host Integration Server Designer UI.

See Also

Other Resources

Creating an Application using Host Integration Server Designer

How to Deploy a Host File Interface

Once you have finished importing and modifying the host file interface, you can deploy the interface. Deploying the interface allows you to write and test code against the interface.

To deploy a Host File interface

1. In HIS Designer, select the tab that has the name of the assembly to deploy.
2. In the **Properties** window, confirm that you have selected the remote environment that you want your assembly to communicate with.
3. In the HIS tree node, right-click on the name of the assembly, and select **Deploy**.

See Also

Other Resources

Creating an Application using Host Integration Server Designer

How to Code a Transaction Integrator Application

Once you have deployed a Transaction Integrator (TI) component, you can write code against that component. Once you are finished writing your code, you can test your code, and if necessary modify the interface to the TI component.

To code a TI application

1. Create an instance of the TI object.

   The TI object contains the interfaces that you will write code against. When your application calls an interface on the TI object, TI Manager will pass the information on to your remote server.

   For more information on creating a TI component and .NET assembly, see **Introduction to COM and COM+ [HIS06]**.

2. Set up your data variables.

   As with many applications that use Host Integration Server, it is important that you use a data type that can successfully translate to and from your remote server. For more information on data types and how they map between systems, see **Data Types [HIS06]** and **Host and Automation Data [HIS06]**.

3. Make calls against any relevant parameters in the TI object.

   Perform any actions necessary to your application, which will likely include calling the interfaces described by your TI object. You may also have additional tasks necessary for your application. For more information, see **Programming Windows-Initiated Processing [HIS06]**.

4. When writing your application, be sure to consider the relevant security details of your environment.

   For more information, see the **Transaction Integrator Security Guide [HIS06]**.

Example

The following example is cut from the main program code from the Discriminated Union tutorial in the SDK sample directory. For the complete code sample, see <Installation Directory>\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\DiscrimiatedUnion.

```
using System;
using System.Collections.Generic;
using System.Text;
using Banking;

namespace DiscriminatedUnions
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Processing Output only Account Information");
            AccountInformationOutOnly();

            Console.WriteLine("\n\nProcessing Input and Output Account Information");
            AccountInformationInOut();

            Console.WriteLine("\nPress any key to continue...");
            Console.Read();

        }

        #region Output Only Discriminated Union Processing
        static void AccountInformationOutOnly()
        {
            // Define an instance of the TI Banking object
            Banking.Accounts MyBankObj = new Banking.Accounts();

            // Call the Get Account Information method on the TI Object
            // passing it the array that contains the checking and saving
            // account information
```

```
            string AccountNumber = "BNK4566112";
            string AccountType = " ";
            Object AcctInfoUnionObj = null;
            string FillerNotUsedByThisSample = " ";

            MyBankObj.GetAInfoOutOnly("111223333", AccountNumber, out AccountType, out Acct
InfoUnionObj, out FillerNotUsedByThisSample);
            switch (AcctInfoUnionObj.GetType().ToString())
            {
                // check the type of the union that was returned to determine
                // whether the array element
                // is a checking or saving account so that the correct
                // structure of the union can be used
                case "Banking.CHECKING":
                        Banking.CHECKING ChkInfo = (Banking.CHECKING)AcctInfoUnionObj;

                        Console.WriteLine("Checking account number: {0}", AccountNumber);
                        Console.WriteLine("\tOverdraft charge:\t {0,10:C2}", ChkInfo.CHK_OD
_CHG);
                        Console.WriteLine("\tOverdraft limit:\t {0,10:C2}", ChkInfo.CHK_OD_
LIMIT);
                        Console.WriteLine("\tLinked account:\t {0,18}", ChkInfo.CHK_OD_LINK
_ACCT);
                        Console.WriteLine("\tLast Statement:\t {0,18}", ChkInfo.CHK_LAST_ST
MT);
                        Console.WriteLine("\tDetail Items:\t {0,18:F0}", ChkInfo.CHK_DETAIL
_ITEMS);
                        Console.WriteLine("\tBalance:\t {0,18:C2}\n", ChkInfo.CHK_BAL);
                    break;

                case "Banking.SAVINGS":
                        Banking.SAVINGS SavInfo = (Banking.SAVINGS)AcctInfoUnionObj;

                        Console.WriteLine("Savings account number: {0}", AccountNumber);
                        Console.WriteLine("\tInterest rate:\t {0,20:P}", SavInfo.SAV_INT_RA
TE / 100);
                        Console.WriteLine("\tService charge:\t {0,18:C2}", SavInfo.SAV_SVC_
CHRG);
                        Console.WriteLine("\tLast Statement:\t {0,18}", SavInfo.SAV_LAST_ST
MT);
                        Console.WriteLine("\tDetail Items:\t {0,18:F0}", SavInfo.SAV_DETAIL
_ITEMS);
                        Console.WriteLine("\tBalance:\t {0,18:C2}\n", SavInfo.SAV_BAL);
                    break;

                default:
                    break;
            }
        }
        #endregion Output Only Discriminated Union Processing
    }
}
```

Optional comments.

How to Test and Modify a Transaction Integrator Application

Once you have finished coding your application, you may test and modify your application. During this process, you may need to undeploy and modify the host file interface.

To test and modify your TI application

1. Before compiling and executing your application, ensure that the host file interface is deployed.
2. Execute your application and test as you would any other application.
3. To undeploy a host file interface, right-click on the name of the .dll that contains the interface and select **UnDeploy**.
4. Make modifications to the interface using HIS Designer, and then save your work.
5. Once you are finished making changes, you can deploy the interface again by clicking on the name of the .dll in HIS Designer and selecting **Deploy**.

See Also

Other Resources

Creating an Application using Host Integration Server Designer

Programming Windows-Initiated Processing

This section discusses various issues you need to understand to program Windows-initiated processing components and applications.

In This Section

Creating a Windows-Initiated Application

In This Section

This section contains the following topics

- How To Confirm that COMTIIntrinsic is Set in Windows XP
- How to Update a Transaction Integrator Assembly
- How to Debug a Visual Basic Application Integration Application
- How To Handle a Host Server Exception

How To Confirm that COMTIIntrinsic is Set in Windows XP

If you use Explicit Security in your application, make sure that the property **COMTIIntrinsic** is set to **–1**. The property **COMTIIntrinsic** is set to **Off** by default for COM+ in Windows XP Professional.

You can use the following Visual Basic script to display and set the **COMTIIntrinsic** value:

```
'Get arguments  ' Set objArgs = WScript.Arguments
 '  if objArgs.Count <> 3 then
 '      WScript.Echo "ComtiSec"
 '      WScript.Echo ""
 '      WScript.Echo "Usage:"
 '      WScript.Echo "ComtiSec [appname], [progid], [value]"
 '      WScript.Echo "[appname]: Name of the application"
 '      WScript.Echo "[progid]:  ProgID of the component to change. Type 'all' for all comp
onents."
 '      WScript.Echo "[value]:   0 for False, 1 for True"
 '      WScript.Quit (0)
 '  end if
   applicationName = "COMTI Utilities"
   componentProgID = "all"
   'ComtiIntrinsics must be set to -1 for comti callback secuity to work.
   ' ComtiIntrinsics = -1 ' This will make Explicit Security work in COM+ on Win XP Pro
   ' ComtiIntrinsics = 0 ' This is the deffault value for COM+ in Win XP Pro
   Set catalog = CreateObject("COMAdmin.COMAdminCatalog.1")
   Set applications = catalog.GetCollection("Applications")
   applications.Populate
   numApplications = applications.Count
   For i = numApplications - 1 To 0 Step -1
       If applications.Item(i).Value("Name") = applicationName Then
           Set application = applications.Item(i)
           Exit For
       End If
   Next
   Set components = applications.GetCollection("Components", application.Value("ID"))
   components.Populate
   numComponents = components.Count
   For i = numComponents - 1 To 0 Step -1
       If components.Item(i).Name = componentProgID Or componentProgID = "all" Then
           Set component = components.Item(i)
           'component.Value("COMTIIntrinsics") = ComtiIntrinsics
      WScript.Echo component.Value("COMTIIntrinsics")
       End If
   Next
   components.SaveChanges
   applications.SaveChanges
   WScript.Echo "Changes All Complete"
```

Note

You must clear the **Protection** and **Disable Changes** properties for the COM+ COMTI Utility application before you run this script.

See Also

Other Resources

Programming Windows-Initiated Processing

How to Update a Transaction Integrator Assembly

If you are upgrading your version of Host Integration Server, you may have Transaction Integrator (TI) assemblies that use a previous version of the .NET Framework. Host Integration Server allows you several options on how to upgrade, so that your assemblies use, so that they may be compatible with the newest version of the .NET Framework.

To update a TI Assembly

1. Do nothing.

   If you do not modify your TI assembly in any way, you do not need to upgrade the assembly: .NET Frameworks 2.0 is backwards-compatible with any TI assembly created using .NET Frameworks 1.x.

2. Open the assembly, make a change, and save the file.

   When you save the file, Host Integration Server will automatically update the TI assembly to the .NET Framework 2.0.

3. Change the name of the assembly in Visual Studio using the **Save As...** command.

   As with option 2, Host Integration Server will update the assembly when you save the name.

How to Debug a Visual Basic Application Integration Application

The following tips will help prevent frustrating debugging sessions:

- When a Transaction Integrator (TI) .NET Framework application is configured to display error numbers (err.number), the number returned is always 0 and not the TI error results. Although TI returns the correct values to COM Interop and COM Interop passes the right values to Visual Basic, Visual Basic considers any positive return code to be success and changes it to 0. To work around this problem, configure the .NET Framework application to return an error description (err.description) instead of the error number. The error description provides accurate and useful error information.

- TI Project parameter type Integer must be defined as a short within Visual Basic.

- TI Project parameter type Long must be defined as an integer within Visual Basic.

- A Visual Basic array index begins at 0, the index of TI parameters defined as arrays starts at position 1. Therefore, it is no longer possible to directly align one for one the index of TI parameters defined as arrays with those defined within Visual Basic.

- Arrays of Decimal data types must be defined as an array of objects, not an array of decimals within Visual Basic. All other arrays of data types can be defined as either an object or data type.

- A common cause of errors during development of host-initiated processing (HIP) .NET Framework components is forgetting to copy all the required assemblies, including all the dependencies, to the HIP Implementing Assemblies folder.

See Also

Other Resources

Programming Windows-Initiated Processing

How To Handle a Host Server Exception

Procedure Title

1.

Procedure Title

1.

Subhead

Insert section body here.

Procedure Title

1.

Example

This is the optional description for a Code Example.

Optional comments.

Compiling the Code

-

Robust Programming

Security

How To Determine Who Initiated a Transaction

It is helpful to be able to determine who initiated a specific transaction, for example, when you need to track down the history of a transaction failure. You can also use this technique to implement resource or transaction-level, per user, security.

When you select either user-level or package-level security on the **Security** tab of the Transaction Integrator (TI) remote environment (RE) properties page, TI sends security information in the session request to the host. If you deploy the Host Account Mapping database known as the Host Account Cache (HAC) and set up a mapping between each Microsoft Windows user and the corresponding host user ID, TI will send that information. Or you can use the **Allow application to override security** option on the **Security** tab, and have the application return any host user ID (and password).

Whether the host does anything with the different user IDs depends mostly on the ATTACHSEC setting for the CICS connection; this corresponds to the APPC LU that TI uses. The default ATTACHSEC setting is **local**, meaning that CICS does not validate the user ID in the session, and CICS runs the transaction in a default host credential. But if you set the ATTACHSEC setting, CICS uses Resource Access Control Facility (RACF) to validate the user ID in the session, and CICS then attaches that user ID to the trusted computing base (TCB) for the transaction as it runs through the mirror transaction into the target mainframe transaction program (TP).

See Also

Other Resources

Programming Windows-Initiated Processing

Managing Security in a Windows-Initiated Application

In This Section

This section includes the following topics

- How to Call a Transaction Integrator Proxy Object in a Secured Virtual Directory
- How To Impersonate Client Application Security Credentials
- How To Use the DPC Security Override

How to Call a Transaction Integrator Proxy Object in a Secured Virtual Directory

One of the interactions Transaction Integrator (TI) has with the windows operating system is the Virtual Directory. In order to use TI and virtual directories together, you need to ensure that you have to correct credentials set on your application. By explicitly using the default credentials for your application, you can ensure that the user's credentials are property replicated across TI and virtual directory.

To call a TI proxy object in a secured virtual directory

1. Create an instance of the new object.
2. Set the credentials of the object to CredentialCache.DefaultCredentials.

   The default credentials of CredentialCache.DefaultCredentials are the credentials of the current user.
3. Continue with your application.

Example

The following code shows how to call a TI proxy object that is inside a secured virtual directory.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;

namespace ELMBankingClient2
{
    class Program

    {
        static void Main(string[] args)
        {
            GetBal.Service MyBal = new ELMBankingClient2.GetBal.Service();
            decimal Balance;
            MyBal.Credentials = CredentialCache.DefaultCredentials;
            Balance = MyBal.GetBalance("Kim Akers", "12345");
            Console.WriteLine(Balance);
        }
    }
}
```

See Also

Other Resources

Programming Windows-Initiated Processing

How To Impersonate Client Application Security Credentials

If you are using Windows-initiated processing (WIP) and are configuring a remote environment (RE) to use Enterprise Single Sign-On (SSO), you have a choice between using the security credentials of the client application or of the COM server program. You make this choice in Transaction Integrator (TI) Manager on the **Security** tab of the ***remote environment*** **Properties** dialog box. If you select **User credentials**, the COM server impersonates the client's security credentials to access SSO and retrieve the host account information. If you select **COM application or ASP.NET credentials**, the COM server uses its own credentials.

Before the COM server can impersonate the client, however, you must program the client to grant the COM server explicit permission to do the impersonating. You can grant the explicit permission in either of two ways:

- Program the client application to call **CoInitializeSecurity(...)** at the beginning of its lifetime, and then request the **Impersonate access level**:

```
HRESULT hr = CoInitializeSecurity(NULL, -1, NULL, NULL,
             RPC_C_AUTHN_LEVEL_CONNECT, RPC_C_IMP_LEVEL_IMPERSONATE
             NULL, NULL, NULL);
```

- Set the default COM security level on your computer to **Impersonate**.

1. Click **Start**, point to **Programs**, point to **Microsoft Host Integration Server 2009**, and then click **TI Manager**.
2. In the TI Manager navigation tree, expand **Component Services**, and then expand **Computers**.
3. Right-click **My Computer**, and then click **Properties**.
4. On the **Default Properties** tab, set the **Default Impersonation Level** to **Impersonate**.
5. Restart your computer so the settings take effect.

The first course of action, adding the **CoInitializeSecurity(...)** call to your application, is the more secure of the two alternatives because it limits the explicit permission to just that client application. The second course of action, changing the default COM security level on your computer, should be chosen only if you are not able to rebuild your client application.

See Also

Reference

Security Tab (Remote Environment Properties)

How To Use the DPC Security Override

Procedure Title

1.

Procedure Title

1.

Subhead

Insert section body here.

Procedure Title

1.

Example

This is the optional description for a Code Example.

Optional comments.

Compiling the Code

-

Robust Programming

Security

Specifying a Remote Environment Programmatically

When an application uses a Transaction Integrator (TI) component, you can structure the application to explicitly specify the remote environment (RE) used by the TI run-time environment. When the application specifies the RE, the application identifies the CICS or IMS region where transaction programs (TP) are executed when they handle method calls to the component. The specific algorithm an application uses to select an RE is up to you. For example, an enterprise can use separate CICS or IMS regions to handle requests from different branches. In this case, the application should set the RE to the appropriate value that identifies the region suitable for the current branch.

In This Section

This section includes the following topics

- How To Use REOverride to Specify a Remote Environment
- Guidelines for Using REOverride

How To Use REOverride to Specify a Remote Environment

To programmatically specify an RE, an application uses the REOverride Context entry. TI Project automatically adds an optional parameter to a TI object for passing TI context data. The application sets the REOverride entry for an object representing a TI component by assigning the name of the RE.

The following Visual Basic code example (with no error checking) shows how to use REOverride to programmatically instruct the TI run-time environment to use a non-default RE named AltREName to handle the calls to Method1, Method2, and Method4. The calls to the three methods are directed to AltREName because the client program has set the context name REOverride to have the value of AltREName and include the optional context parameter ContextArray. The call to Method3 is directed to the RE assigned as default to the object because it does not include the optional ContextArray context parameter.

```
Dim Obj As Object
Dim ConObj AS Object
Dim ContextArray() As Variant

Set ConObj = CreateObject("COMTI.ContextObject")
Set Obj = CreateObject("Your.Object")

ConObj.WriteContext "REOverride", "AltREName", ContextArray

Obj.Method1 parm1, parm2, parm3, ContextArray
Obj.Method2 parm1, parm2, ContextArray
Obj.Method3 parm1, parm2, parm3, parm4, parm5
Obj.Method4 parm1, parm2, parm3, parm4, ContextArray
```

See Also

Tasks

Specifying a Remote Environment Programmatically

Guidelines for Using REOverride

Use the following guidelines for when to use REOverride to set an RE programmatically:

- Avoid hard-coding RE names into applications. Instead, load RE names from a file or database.
- Ensure that applications are structured to handle failures when they attempt to set the RE.
- Structure your application code to use a list of RE names. This practice reduces errors caused by missing REs.
- Ensure that procedures for adding and configuring REs include a mechanism to update REs referenced in the application code.

Note
Use REOverride to confirm that administrative and operational tasks do not interfere with application code that sets an RE. S pecifically, review when and how REs are deactivated and deleted.

To implement RE selection by using REOverride, you must ensure that the TI component starts out with an associated RE instance even though the application will set the RE programmatically. The RE that is currently assigned to the component is used when an application does not explicitly set the RE.

See Also

Tasks

Specifying a Remote Environment Programmatically

How to Program with Discriminated Unions

A discriminated union is a data structure that can hold a data value of several different types. Host Integration Server uses discriminated unions with several providers, such as the Managed Provider for Host Files. When creating an application that uses Remoting or Web Services, you must satisfy the Web Services Description Language (WSDL) requirements for the discriminated union. WSDL generation constraints require that all structures in an object be used in a method call. Therefore, you need to ensure that all the structures in a discriminated union are also used, even if only in a piece of stub code.

To use a discriminated union with Remoting or Web Services

1. Create your schema as normal.
2. Identify any structure in the discriminated union that is not explicitly used in another method call.
3. Create a dummy method call that calls the unused structure.

Example

The following example shows a line of dummy method that uses several discriminated union structures. By having such a method, the WSDL generation requirements are satisfied.

```
void dummyroutine1 (ACCT_TYPE_SAVE acct_type_sav, ACCT_TYPE_CHK acct_type_chk)
```

See Also

Other Resources

Programming Windows-Initiated Processing

How To Override Settings in the Type Library

If you need to temporarily change certain data sent from the type library to the host, you can override the type library settings without changing the original file. Use the following keywords in combination with the COMTIContext parameter to override the type library setting:

- CONNTIMEOUT
- CONNTYPE
- IMS_LTERM
- IMS_MODNAME
- LibNameOverride
- OverrideSourceTP
- PASSWORD
- PortOverride
- ProgNameOverride
- RecvTimeOut
- REOverride
- SendTimeOut
- TPNameOverride
- USERID

The override remains active until either a new override is set or the override is deleted. Use the **WriteContext** function to set the override and the **DeleteContext** function to delete the override. If you delete the override, the value defaults to the setting in the original type library.

See Also
Tasks
How to Pass a Custom TRM
Concepts
Using Custom TRMs and ELMs with COMTIContext

Using a Persistent Connection

Windows-initiated processing (WIP) supports persistent connections over TCP/IP and SNA for the following programming models:

- IMS Connect
- TCP Transaction Request Message (TRM) Link
- TCP Enhanced Listener Message (ELM) Link
- TCP Transaction Request Message (TRM) User Data
- TCP Enhanced Listener Message (ELM) User Data
- OS/400 DPC
- CICS Link LU 6.2
- CICS User Data LU 6.2

Persistent connections are not supported in the following programming models:

- IMS Implicit
- IMS Explicit
- IMS LU 6.2

Windows-initiated processing (WIP) persistent connections allow you to maintain a single TCP connection or SNA conversation over multiple method calls to the host. In Host Integration Server 2000, COMTI had to open and close a connection each time a method call was made to the host. On the mainframe side, CICS had to start and stop a transaction program (TP). In Host Integration Server 2009, persistent connections allow Transaction Integrator (TI) to open a connection for the first method in a group of methods, make all the method calls, and then close the connection. On the mainframe side, CICS starts an instance of the transaction program, keeps the instance active between method calls, and then stops the program after the last call.

One of the major benefits from using persistent connections is that it allows CICS to maintain state across multiple method calls and allows for the use of local variables. Persistent connections are implemented and managed through the COMTIContext.

COMTIContext supports methods that flow to the COM+ or .NET Framework application and updates client status information (**COMTIContext** array) or closes persistent connections.

UpdateContextInfo updates the clients **COMTIContext** array with information obtained from the COM+ or .NET Framework application object, but with no server object involvement.

ClosePersistentConnection closes persistent connections by contacting the COM+ or .NET Framework application object, but with no server object involvement.

The client can obtain connection state information by calling the **GetConnectionInfo** method that is implemented by the **COMTIContext** object. In the case of a .NET Framework method failure, the client must call **UpdateContextInfo** before it calls **GetConnectionInfo**.

A time-out mechanism reclaims orphaned persistent connections. The new **COMTIContext** keyword **CONNTIMEOUT** takes an integer value specifying, in seconds, how much time elapses before a persistent connection is considered abandoned, and then automatically closed. The timing starts as the client call processing is completed by the COM+ or .NET Framework generic object.

GetConnectionInfo querys the status of a persistent connection. The following shows a COM-based method:

```
HRESULT GetConnectionInfo (
     [in, out] SAFEARRAY(VARIANT)*COMTIContextArray,
     [out] BOOL* pfConnectionIsPersistent,
     [out] BOOL* pfConnectionIsViable);
```

The following shows the .NET Framework-based equivalent:

```
GetConnectionInfo (ref object[] contextArray,
    out bool fConnectionIsPersistent,
    out bool fConnectionIsViable).
```

The **COMTIContextArray** parameter is updated to reflect the state of the connection, the **pfConnectionIsPersistent** parameter contains TRUE if the connection is persistent and active, and the **pfConnectionIsViable** parameter contains TRUE if the connection is active.

UpdateContextInfo updates the clients **COMTIContext** array. The following shows a COM-based method:

```
HRESULT UpdateContextInfo (
    [in, out] SAFEARRAY(VARIANT)*COMTIContextArray);
```

The following shows the .NET Framework-based equivalent:

```
UpdateContextInfo (ref object[] contextArray).
```

The **COMTIContextArray** parameter is updated to reflect the state of the connection. At a later time other information kept in the COM+ or .NET Framework application might also be returned in the update **COMTIContextArray**.

ClosePersistentConnection closes a persistent connection without the need for a call to the server system. The following shows a COM-based method:

```
HRESULT ClosePersistentConnection (
    [in, out] SAFEARRAY(VARIANT)*COMTIContextArray);
```

The following shows the .NET Framework-based equivalent:

```
ClosePersistentConnection (ref object[]COMTIContextArray).
```

The **COMTIContextArray** parameter is updated to reflect the state of the connection.

In This Section

- About Persistent Connections
- Programming Models that Support Persistent Connections
- How To Use a Persistent Connection

About Persistent Connections

Insert introduction here.

Subhead

Insert section body here.

Subhead

Insert section body here.

Programming Models that Support Persistent Connections

Insert introduction here.

Subhead

Insert section body here.

Subhead

Insert section body here.

How To Use a Persistent Connection

The following topic describes how to use a persistent connection with Windows-Initiated Processing (WIP)

To use a persistent connection with WIP

1. Set the COMTIContext keyword CONNTYPE to OPEN.

   If a call with CONNTYPE set to OPEN completes successfully, the returned COMTIContext array CONNTYPE keyword will have a value of USE.

   After you set the COMTIContext keyword CONNTYPE to OPEN, you can choose to set CONNTYPE to USE. However, this action is not mandatory because it is set to USE by default.

2. Once you have established the connection, you can use the COMTIContext object to access the mainframe.

3. If the method call fails, use UpdateContextInfo and GetConnectionInfo on the COMTIContextLib.ContextObject to obtain updated status of the connection.

4. To make a call and terminate the persistent connection, set the CONNTYPE keyword to CLOSE.

   If the call completes successfully, the returned COMTIContext array CONNTYPE keyword will have a value of NON-PERSISTENT.

   Optionally, you can call ClosePersistentConnection at any time to close a persistent connection. The connection will be terminated and there will be no interaction with a server program.

Example

The following Visual Basic 6.0 code example shows how to use the OPEN and CLOSE method calls that might return an error. The sample also demonstrates how to determine whether a connection can still be used.

```
Public CtxCount As Long
Public COMTIContext() As Variant
Public ContextObj As COMTIContextLib.ContextObject

Dim fConIsPersistent as Boolean
Dim fConnIsViable as Boolean
Dim varConnType as Variant

Private Sub cmdBalance_Click()
  On Error GoTo ErrorHandler

OpenCall:
  varConnType = "OPEN"
  ContextObj.WriteContext "CONNTYPE", varConnType, COMTIContext
  lngReturn = objBank.cedrbank(txtName.Text, txtAccount.Text, curRetBalance, COMTIContext)

UseCall:
  lngReturn = objBank.cedrbank(txtName.Text, txtAccount.Text, curRetBalance, COMTIContext)

CloseCall:
  If (fCloseWithMethod) Then
      varConnType = "CLOSE"
      ContextObj.WriteContext "CONNTYPE", varConnType, COMTIContext
      lngReturn = objBank.cedrbank(txtName.Text, txtAccount.Text, curRetBalance, COMTIConte
xt)
  Else
      COMTIContext = objBank.ClosePersistentConnection
  End-if

  Exit Sub

ErrorHandler:
    COMTIContext  = objBank.UpdateContextInfo   Optional for COM required for .NET
    ContextObj.GetConnectionInfo (COMTIContext, fConnIsPersistent, fConnIsViable)
    If (fConnIsPersistent = True And fConnIsViable = True) Then
```

```
        Continue with the next Use or Close method call is OK
    Else
        Connection is either Non-persistent or no longer viable
        So a Use or Close call is not valid
    End-if
    Exit Sub

End Sub
```

See Also

Other Resources

Using a Persistent Connection

COMTIContext Interface

COMTIContext Keywords

How to Self-Host a Windows-Initiated Process

Self-Hosting is a technology that allows you the option of running a Transaction Integrator (TI) assembly in-process with an associated application. Self-Hosting improves performance of your application by not running the TI assembly through Internet Information Services (IIS).

To self-host a TI assembly

1. Create a TI assembly as you would normally.

   For more information, see Creating an Application using Host Integration Server Designer.

2. Register the TI assembly using TI Manager, using the **Self-Host** radio button selected.

   You may also set the hosting model in the Properties toolbox in Visual Studio.

   For more information, see Creating an Object.

3. In Visual Studio, use Solution Explorer to add a reference to your TI assembly.

4. Code your application, using the interfaces on the TI assembly as you would any other interface.

See Also

Other Resources

Programming Windows-Initiated Processing

How To Verify a Remote Installation

Procedure Title

1.

Procedure Title

1.

Subhead

Insert section body here.

Procedure Title

1.

Example

This is the optional description for a Code Example.

Optional comments.

Compiling the Code

-

Robust Programming

Security

Programming Host-Initiated Processing

This section discusses various issues you need to understand to program host-initiated processing components and applications.

In This Section

- Connecting HIP Components to Visual Basic Applications

How To Connect a HIP Component to a Visual Basic Applications

A key link in host-initiated processing (HIP) is the connection between the HIP component in Transaction Integrator (TI) and the client application. The connection between the HIP component and the Microsoft Visual Basic server DLL is created through matching elements of the HIP type library with elements of the Visual Basic project. The following table shows the relationship between the elements.

HIP Type Library	Visual Basic Project
Type Library name	Visual Basic Project name.
Type Library Interface name	Visual Basic Class name.
Type Library Method names	Functions or subroutines within the Visual Basic Class.
Type Library method parameters	Defined one for one within the Visual Basic functions or subroutines with the Visual Basic subroutines and functions.

Note
Make sure that the Visual Basic server DLL is registered.

If you are using the **Implements** key word within a Visual Basic server, the following additional rules apply:

- The **Visual Basic Implement Compatible Interface** property must be enabled. Set the property on the component *interface* **Properties** page within the TI type library or assembly
- All parameters defined to type library methods must be **Input\Output**. The **Implements** keyword does not support parameters defined as either input or output. All parameters must be defined as input or output.
- The function or subroutine calls must be defined as public, not private, within the Visual Basic class.

See Also

Other Resources

Programming Host-Initiated Processing

How to Use a Persistent Connection with Host-Initiated Processing

A persistent connection is a connection that stays open past the duration of a specific call. Because your application does not need to re-create the connection on each call, you can use a persistent connection to increase the efficiency of your Host-initiated application. An application that uses a persistent connection with Host-initiated Processing (HIP) operates in many ways the same way as a Windows-Initiated Processing (WIP). The difference, of course, is that the mainframe initiates and terminates the connection, while the windows application responds to the requests of the mainframe.

Note
Host Integration Server supports many of the same programming environments for HIP as for WIP. The exceptions are IMS Connect, Distributed Program Call (DPC), and SNALink, which are not supported for HIP persistent connections.

To use a persistent connection with HIP

1. Receive a call with your Windows application from the mainframe, indicating that a connection has been created.

   It is the responsibility of the mainframe application to request the persistent connection.

2. Have your Windows application react to the request in the relevant manner.

   There is nothing specific your application must do in order to use a persistent connection: creating and terminating the connection is the responsibility of the mainframe application.

3. Optionally, you can create a new instance of the HIPServerUserContext to query the status of the connection.

   The new instance is automatically created with the context information for the relevant connection. Using HIPServerUserConext, you can determine what type of connection the mainframe has created, and react accordingly.

Example

The following code is pulled from the CICS sample application in the SDK. The sample uses the CONNTYPE of the server object to perform different actions.

```
decimal GetAccountBalance(object[] contextArray)
        {
            decimal ReturnBalance = 0.0m;
            string ConnType;
            object contextValue;

            _TIServerContext.ReadContext("CONNTYPE", out contextValue, ref contextArray);

            if (contextValue == null)
                ReturnBalance = 123.45m;
            else
            {
                ConnType = contextValue.ToString();
                ConnType.ToUpper();
                switch (ConnType)
                {
                    case "OPEN":
  // Set the initial value of the Account Balance
  // and save it in a global varaible and return it.
                        ReturnBalance = 123.45m;
                        _AccountBalance = ReturnBalance;
                        break;

                    case "USE":
  // Increase the value of the global Account Balance
  // varaible and and return its value. Save this new value
  // in the global variable for later use
                        _AccountBalance += 100;
                        ReturnBalance = _AccountBalance;
                        break;
```

```
                    case "CLOSE":
    // Increase the value of the global Account Balance
    // variable and return the new value. Set the global variable
    // to zero because the "CLOSE" call indicates that we are
    // done with it.
                        ReturnBalance = _AccountBalance + 150;
                        _AccountBalance = 0.0m;
                        break;

                    case "UNKNOWN":
                    default:
                        _AccountBalance = 0.0m;
                        ReturnBalance = 123.45m;
                        break;
                }
            }

            return ReturnBalance;
        }
```

The code sample uses a global variable to store information. It is also possible to use the context object itself to store information. Although not shown here, it is possible to use the context object to pass information back to the windows application.

See Also

Concepts

CICS Sample

Other Resources

Using a Persistent Connection

Application Integration Security Guide

This section provides information about steps you can take to safeguard Transaction Integrator, your data, and your network when you are programming client or server applications.

In This Section

Mainframe Authentication for CICS LINK

Resource-level authentication is recommended in the CICS region. Due to a restriction imposed by the IBM distributed program link (DPL) protocol, a user ID and password transmitted from the workstation by Transaction Integrator (TI) are ignored and not used for transaction-level authentication. The target CICS region expects, under such circumstances, that authentication has been completed by the application that executes the IBM DPL; for example, a TI application on the PC. (Traditionally, the application that executes an IBM DPL has been another CICS region.)

Instead, for transaction-level authentication, the target CICS region associates the default user ID for the region with the transaction ID of the CICS (Mirror transaction) task and the user ID from the sender is ignored. Unless this practice is taken into consideration, attempts to secure the Mirror transaction can cause an application malfunction because of the failure to authenticate.

See Also

Concepts

AS/400 Security

Limitations of User Access Level Sign On

Using SSO with Host-Initiated Processing

Using SSO with Encrypted Passwords

Other Resources

Threat Mitigation within Visual Studio

Application Integration Security Guide

AS/400 Security

The support for AS/400 security is the same as for other Windows-initiated operations against the mainframe, with the following adjustments:

- No support for RACF, AFC/2, Kerberos, or Top Secret
- Integration with AS/400 native security system only
- Support for Single Sign-On through SSO
- Support for SSL security

See Also

Other Resources

Application Integration Security Guide

Limitations of User Access Level Sign On

When you sign on with only user access permissions, you have restricted capabilities for using Transaction Integrator (TI). In Visual Studio, user access enables you to do the following:

- Open TI projects
- Create and save new type libraries
- Open and save existing type libraries

See Also

Other Resources

Application Integration Security Guide

Using SSO with Host-Initiated Processing

When you use Single Sign-On (SSO) security with host-initiated processing (HIP), the impersonation of user credentials is handled differently depending upon whether you are calling a .NET object or a COM object. If HIP is calling a .NET object, there are no special considerations; the Transaction Integrator (TI) run-time environment impersonates the user account. If HIP is calling a COM object, however, there are special considerations.

Depending on the threading model and registration type of the components, the following actions occur when HIP calls the method of a .COM object when it is impersonating a user account (the one that the host credentials would have been mapped to via SSO):

Threading Model	Registration	Actions
Single, apartment	Server/Library/No COM+ application	• Server object methods are called under HIP Service/COM+ application configured identity. • Server object methods call **CoImpersonateClient** to start impersonating the user identity. • Optionally, methods can call **CoRevertToSelf**, although that is not necessary because COM will call it anyway after the method returns.
Free, both, neutral	Server COM+ application	• Server object methods are called under HIP Service/COM+ application configured identity. • Server object methods call **CoImpersonateClient** to start impersonating the user identity. • Optionally, methods can call **CoRevertToSelf**, although that is not necessary because COM will call it anyway after the method returns.
Free, both, neutral	Library/No COM+ application	• Server object methods are called under the user identity being impersonated. • Server object method should not call **CoImpersonateClient** or **CoRevertToSelf** because they would fail with RPC_E_CALL_COMPLETE.

If you are programming in Microsoft Visual Basic® 6.0, be sure to include the **CoImpersonateClient** and **CoRevertToSelf** declarations in your programs:

```
Private Declare Function CoImpersonateClient Lib "ole32.dll" () As Long
Private Declare Function CoRevertToSelf Lib "ole32.dll" () As Long
```

See Also
Concepts
Mainframe Authentication for CICS LINK
AS/400 Security
Limitations of User Access Level Sign On
Using SSO with Encrypted Passwords
Other Resources
Threat Mitigation within Visual Studio
Application Integration Security Guide

Using SSO with Encrypted Passwords

If you are using Single Sign-On (SSO) security, you must determine whether the client application passes plain text passwords or encrypted passwords to the Transaction Integrator (TI) run-time environment. If the passwords are in plain text, SSO accepts the passwords when they are submitted by the TI run-time environment. If the passwords are encrypted, SSO does not accept the passwords, and the call to SSO fails. To avoid failed SSO calls, disable password validation for the SSO affiliate application.

Threat Mitigation within Visual Studio

Product security is a top priority throughout Microsoft development. Beginning with the Microsoft Windows Security Push in 2002, Microsoft has invested additional time and resources to developing more secure code and detailed instructions for deploying and securing your computing environment.

The Host Integration Server product team conducted a complete threat modeling analysis to identify and mitigate potential areas of concern. A threat model is a security-based analysis that helps you determine the highest-level security risks posed to a product or application and how attacks can manifest themselves.

Although Microsoft has mitigated all known internal security threats to Host Integration Server, you should take steps to mitigate threats from elsewhere in your network environment. Threat modeling helps you evaluate the threats to the applications you are writing or running, and thereby reduce the overall risk to your computer system. For more information about threat model analysis, see Chapter 4 Threat Modeling in Michael Howard and David LeBlanc, *Writing Secure Code 2nd Edition*, Redmond, WA: Microsoft Press. 2003.

Howard and LeBlanc summarize six categories of possible security threats to your computing environment:

- **Spoofing identity**. Spoofing threats enable an attacker to pose as another user or enable a rogue server to pose as a valid server. An example of user identity spoofing is illegally gaining access and then using another users authentication information, such as username and password.

- **Tampering with data**. Data tampering involves malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.

- **Repudiation**. Repudiation threats are associated with a user who denies that he performed an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations.

- **Information disclosure**. Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, a users ability to read a file that she was not granted access to and an intruders ability to read data in transit between two computers.

- **Denial of service**. Denial of service (DoS) attacks deny service to valid users—for example, by making a Web server temporarily unavailable or unusable. You protect against certain types of DoS threats simply to improve system availability and reliability.

- **Elevation of privileges**. In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.

Howard and LeBlanc also point out that some threat types can interrelate. For example, it is possible for information disclosure threats to lead to spoofing threats if the users credentials are not secured. Similarly, elevation of privilege threats is by far the worst because if someone can become an administrator or root on the target computer, every other threat category becomes a reality. Conversely, spoofing threats might lead to a situation where escalation is no longer needed for an attacker to achieve his goal.

To mitigate threats that originate outside Transaction Integrator (TI) but which can negatively affect TI components and your application, Microsoft recommends that you do the following:

- Protect the TI COM Type Library or .NET Assembly from Unauthorized Access

- Protect the Output from Tracing and Network Monitoring Activities

- Protect the TI Record or Playback Files from Unauthorized Access

Protecting the TI COM Type Library or .NET Assembly from Unauthorized Access

To prevent an attacker from viewing or modifying the contents of a Transaction Integrator (TI) COM type library or .NET assembly and then using that information to either create a client application which spoofs the identity of an authorized user or modify the custom properties of the component, you should:

- Place the computer running Visual Studio and TI Designer in a secure location.
- Confirm that the access permissions to Visual Studio, TI Designer, or any other tool used to modify TI type libraries and .NET assemblies are set correctly.
- Store all TI component type libraries and .NET assemblies in a secure directory.
- Confirm that the access permissions are set correctly on all type libraries and .NET assemblies.
- Confirm that the access permissions are set correctly on the directory that contains the type libraries and .NET assemblies.

See Also

Concepts

Protecting the Output from Tracing and Network Monitoring Activities

Protecting the TI Record or Playback Files from Unauthorized Access

Other Resources

Threat Mitigation within Visual Studio

Protecting the Output from Tracing and Network Monitoring Activities

To prevent an attacker from viewing the user credential information that might be stored in trace files or network monitoring files, you should:

- Confirm that only authorized users are allowed to run the SNA TRACE or Microsoft Network Monitoring programs on the computer that is running Transaction Integrator.
- Store all tracing (Tracebits) output files and network monitoring (Netmon) output files in a secure directory.
- Confirm that the access permissions are set correctly on all tracing output files and network monitoring output files.
- Confirm that the access permissions are set correctly on the directory that contains the output files.
- Delete all tracing output files and network monitoring output files as soon as you are done with them.

See Also
Concepts
Protecting the TI COM Type Library or .NET Assembly from Unauthorized Access
Protecting the TI Record or Playback Files from Unauthorized Access
Other Resources
Threat Mitigation within Visual Studio

Protecting the TI Record or Playback Files from Unauthorized Access

To prevent an attacker from either viewing the contents of the Transaction Integrator (TI) record or playback files or replacing those files with ones that could record user data sent to or from the host, you should:

- Store all TI record and playback files in a secure directory.
- Confirm that the access permissions on all record or playback files are set correctly. A user must have administrator rights to be able to record and save a file.
- Confirm that the access permissions are set correctly on the directory that contains the record or playback files are stored.
- Store the record file in a form other than plain text.

See Also

Concepts

Protecting the TI COM Type Library or .NET Assembly from Unauthorized Access

Protecting the Output from Tracing and Network Monitoring Activities

Other Resources

Threat Mitigation within Visual Studio

Data Integration Programmer's Guide

This section provides information required to develop applications to access data in an environment using Host Integration Server 2009. This section provides documentation for developers about data access, data replication, and data tools.

For API reference and other technical information about data integration, see the Data Integration Programmer's Reference section of the SDK.

For sample code that illustrates data integration, see the Data Integration Samples section of the SDK.

For more information, see the Data Integration User's Guide in the Operations section.

In This Section

Introduction to the Data Integration Programmer's Guide

Data Integration refers to the set of tools and techniques you can use from a Windows environment to access and manipulate database information on a remote host system. This section provides information required to develop applications to access data in that environment using Microsoft Host Integration Server 2009.

In This Section

- Supported Data Integration Programming Scenarios
- What You Need to Know to Program Data Integration
- Additional Resources for Data Integration Programming

Supported Data Integration Programming Scenarios

You can develop applications for data integration used in a Host Integration Server environment using several different development tools and application programming interfaces. The following table describes the different programming tools available, and the different languages usable for each tool.

Language	Managed provider	Unmanaged provider	ODBC provider
Visual Basic	Managed Provider for DB2 using managed Visual Basic Managed Provider for Host files using managed Visual Basic	ADO Provider for AS/400 and VASM ADO Provider for DB2 Host File Transfer ActiveX controls for MVS, OS/390, AS/400, and AS/36 Data Queue ActiveX for AS/400 Data Queues	ADO to access DB2 using ODBC
C/C++	Managed Provider for DB2 using managed C/C++ Managed Provider for Host files using managed C/C++	OLE DB Provider for AS/400 and VASM OLE DB Provider for DB2 Host File Transfer ActiveX controls for MVS, OS/390, AS/400, and AS/36 Data Queue ActiveX for AS/400 Data Queues	ODBC Driver for DB2
.NET Frameworks (C++, C#, VB.NET)	Managed Provider for DB2 Managed Provider for Host Files		

See Also

Other Resources

Introduction to the Data Integration Programmer's Guide

What You Need to Know to Program Data Integration

To use this section effectively, you should be familiar with:

- Host Integration Server 2009
- One of the following operating environments:
    - Microsoft Windows Server 2003
    - Microsoft Windows XP
    - Microsoft Windows 2000 Server
- SNA concepts

Depending on the application programming interface and development tools used, you should be familiar with:

- Microsoft COM objects
- Microsoft OLE DB
- Microsoft ADO
- Microsoft ODBC
- Microsoft .NET

See Also

Other Resources

Introduction to the Data Integration Programmer's Guide

Additional Resources for Data Integration Programming

This section does not describe the products, architectures, or standards developed by other companies or organizations.

For information about SNA architecture, see your system network documentation.

The following documents provide additional information about the OLE DB application programming interfaces (APIs):

- Microsoft® Data Access Components (MDAC) Software Development Kit 2.8

The following documents provide additional information about Microsoft ActiveX® Data Objects:

- Microsoft Data Access Components (MDAC) Software Development Kit 2.5

The following documents and publications provide additional information about the Open Database Connectivity (ODBC) standard and ODBC programming:

- Microsoft Data Access Components (MDAC) Software Development Kit 2.5
- Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference
- *Inside ODBC*,written by Kyle Geiger and published by Microsoft Press

For more information about SNA and the Distributed Data Manager (DDM), see the following manuals:

- *IBM Distributed Data Management Architecture: General Information* (Document Number GC219527-3)
- *IBM OS400 Distributed Data Manager User's Guide*
- *IBM OS400 Distributed Data Manager Programmer's Guide*
- *IBM Systems Network Architecture: Technical Overview*
- *IBM Systems Network Architecture: Concepts and Products*
- *IBM SNA Format and Protocol Reference Manual: Architectural Logic*
- *IBM DFSMS/MVS Version 1 Release 2 DFM/MVS Guide and Reference* (Document Number SC26-4915-00)
- *IBM DFSMS/MVS Version 1 Release 3 DFM/MVS Guide and Reference* (Document Number SC26-4915-01)
- *IBM DFSMS/MVS Version 1 Release 4 DFM/MVS Guide and Reference* (Document Number SC26-4915-02)

For more information about IBM DB2, see the following manuals:

- *IBM DB2 for OS/390 Version 5 Reference for Remote DRDA: Requesters and Servers* (Document Number SC26-8964-00)
- *IBM DB2 for OS/390 Version 5 Application Programming and SQL Guide* (Document Number SC26-8958-00)
- *IBM DATABASE 2 Administration Guide for Common Servers Reference* (Document Number S20H-4580)
- *IBM DATABASE 2 Application Programming Guide for Common Servers Reference* (Document Number S20H-4643)
- *IBM DB2 Universal Database API Reference* (Document Number S10J-8167)

- *IBM DB2 Universal Database Building Applications for Windows and OS/2 Environments Reference* (Document Number S10J-8160)

For background information about logical unit (LU) 6.2, Advanced Program-to-Program Communications (APPC), or the Common Programming Interface for Communications (CPI-C), see the following manuals:

- *IBM SNA: Technical Overview*
- *IBM SNA: Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM SNA: Formats*
- *IBM Systems Network Architecture: Introduction to APPC*
- *IBM Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*

See also:

Introduction to Data Integration

Data Integration User's Guide

Data Access Library Programmer's Guide

This section contains information describing how to access the capabilities of the Data Access Tool using the interface exposed by the Data Access Library.

For API references and other technical information about the Data Access Library, see the Data Access Library Programmer's Reference section of the SDK.

For information about how to use the user interface (UI) of the Data Access Tool, see the Data Access Tool section in the Operations guide.

In This Section

Data Access Library

Programming with the Data Access Library

Data Access Library

This section describes the background for the Data Access Library (DAL), how the different interfaces relate to each other, and what technologies you must be familiar with to program the DAL.

In This Section

Goals for the Data Access Library

The original goal of the Data Access Library (DAL) was to expose an interface that would enable developers to automate lengthy tasks normally performed through the Data Access Tool (DAT) user interface. As development progressed, the Data Access Library grew to include the most common functionality of the DAT.

When Microsoft developers originally created the Data Access Tool, they realized that some tasks were relatively time-consuming. For example, creating a DB2 data package takes several steps through the DAT user interface. The developers determined that the best way to reduce the time to perform such tasks was to expose a programming interface to the DAT. They determined that in exposing the interface that is required to automate a lengthy procedure, they could expose essentially the entire functionality of the Data Access Tool.

Therefore, the purpose of the Data Access Library interface is to provide a programmatic interface for the Data Access Tool and Data Source Wizard. This is done through a straightforward interface that correlates almost entirely to actions that you can take using the appropriate user interface. Although the Data Access Library provides a subset of the capabilities the user interface exposes, you can automate most of the common tasks for which you would typically use the UI.

See Also
Other Resources
Data Access Library

Data Access Library Interface

The Data Access Library is based around four objects that represent the core functionality of the Data Access Tool (DAT) and Data Source Wizard. The following table describes the core objects for the Data Access Library.

Object	Description
DataAccessControl	Provides access to the options listed on the DAT Action menu: testing connections and queries, creating packages, and converting data.
DataAccessSettings	Provides access to the functionality of the Data Source Wizard: describing where information is stored, and controlling the Wizard dialog boxes and warnings.
IConnectionString	Base class for describing the different types of connection strings that the DAT uses.
IHCDItem	Base class for describing the objects that make up a host column description (HCD) file.

In addition to the above objects, the Data Access Library interface has an exception object, possible data types, a callback function triggered when creating packages, and a collection of IHCD items. For more information, see the Data Access Library Programmer's Reference.

See Also

Other Resources

Data Access Library

What You Should Know Before Programming the Data Access Library

The Data Access Library (DAL) is a .NET Framework interface for creating universal data link (.udl) files for Host Integration Server. Therefore, you should be familiar with the following concepts and technologies:

- Host Integration Server 2009

  The purpose of the DAL is make it easier to create .udl files, which Host Integration Server then uses to access remote servers through a variety of technologies. Therefore, you should be familiar with the general framework that you will be working with when you use the DAL.

- Data Access Tool (DAT) and Data Access Wizard

  The DAT and the Data Access Wizard are the technologies that enable you to create .udl files and ODBC DSNs through a user interface. Therefore, you should be familiar with the standard way in which the technology performs a task before you try to perform that task programmatically.

- .NET Framework 2.0

  The Data Access Library was written using Microsoft Visual Studio 2005 and the .NET Framework 2.0. Therefore, in order to successfully use the DAL, you should be familiar with the technologies that went into constructing it.

See Also
Other Resources
Data Access Library

Supported Platforms for the Data Access Library

As a programmatic interface to the Data Access Tool (DAT), the Data Access Library (DAL) is supported by all systems that also support the DAT.

See Also

Other Resources

Data Access Library

Programming with the Data Access Library

The main programming tasks for the Data Access Library (DAT) can be considered one of three tasks: creating a connection string, retrieving or modifying data in a connection string, or performing an administrative task.

In This Section

Creating a Connection String

How to Retrieve Data

Performing Administrative Tasks

See Also

Other Resources

Data Access Library

Creating a Connection String

The Data Access Tool can create one of two basic types of connection strings: an OLE DB connection string stored in a universal data link (.udl) file, and an ODBC connection string stored in a data source name (DSN).

For information about how to view a connection string using the UI, see How to Display an Initialization String in the Operations guide.

In This Section

How to Create a Connection String for a .udl File

How to Create a Connection String for an ODBC System, User or File DSN

See Also

Other Resources

Programming with the Data Access Library

How to Create a Connection String for a .udl File

A universal data link (.udl) file is essentially a text file that contains the connection string for an OLE DB data source. You can create a .udl file by using the appropriate **DB2OleDbConnectionString** or **FileSysOleDbConnectionString** constructor, and then save the string to secondary storage with a call to **Save**. The Data Access Library automatically creates the appropriate .udl file to store the string in, and save the file to disk.

To create a .udl file and associated connection string

1. Create an empty connection string by calling a connection string constructor.

   Calling the constructor creates a connection string with default settings. These default settings can be set only through the Data Access Tool user interface.

   If you use a file path for a file that currently exists, the system loads the connection string information in that file instead.

   You can determine the default path your system uses for storing .udl files with a call to **DataAccessSettings.MakeUDLPath**. **DataAccessSettings** also stores the default paths for DSN and HCD files.

2. Add in the relevant connection information to the connection string by calling the various connection string properties, such as **DataSourceName**, **UserName**, or **Password**.

   You can also retrieve the full connection string as a text string with a call to **GetString**, and then save the modified string with **SetString**.

3. Save the string by calling the relevant **Save** method, such as **DB2OleDbConnectionString.Save**.

   The system saves the connection string in a .udl file. The system creates the .udl file using the file path passed in the *name* parameter of the constructor. If the file does not contain the full path, the system uses the default path as described in **DataAccessSettings.UDLpath**.

The following code example demonstrates how to create a .udl file using a new file name, user name, and password.

```
static DB2OleDbConnectionString CreateUDLFile(string FileName, string NameOfUser, string Pa
ssWord, ref System.Exception myException)
{
   try
   {
      DB2OleDbConnectionString myConnection = new DB2OleDbConnectionString(FileName, false)
;
      myConnection.UserName = NameOfUser;
      myConnection.Password = PassWord;
      myConnection.Save();
      System.Exception MyEx= new System.Exception(@"Successful Creation", null);
      myException = MyEx;
      return myConnection;

   }
   catch (Exception ex)
   {
      myException = ex;
      return null;
   }
}
```

See Also

Tasks

How to Retrieve Data

How to Edit a Configuration

Other Resources

Programming with the Data Access Library

How to Create a Connection String for an ODBC System, User or File DSN

The other type of data that you can create by using the Data Access Library is a data source name (DSN). Like a universal data link (UDL) file, a DSN contains the connection information necessary to access a remote database. However, where a .udl file is generally used for an OLE DB database connection, most systems use DSN files for ODBC connections. Further, a DSN does not necessarily have to be a file: a DSN can be a file, a registry-based system DSN, or a registry-based user DSN.

To create a DSN file and associated connection string

1. Call the constructor for **DB2OdbcFileConnectionString**, **DB2OdbcSysConnectionString**, or **DB2OdbcUserConnectionString**.

   Calling the constructor creates a connection string with default settings. These default settings can be set only through the Data Access Tool user interface.

   If you use a file path for a file that currently exists, the system loads the connection string information in that file instead.

2. Fill in the relevant connection string properties by calling the properties of the object created.

   You can also fill in the connection string with **GetString**, which returns the connection string as a text string. After you finish modifying the relevant values, you can return the connection string to the object with a call to **SetString**.

3. Save the connection string information back into storage with a call to **Save**.

The following code example demonstrates how to create a DSN file and associated connection string.

```
static DB2OdbcFileConnectionString CreateUDLFile2(string FileName, string NameOfUser, strin
g PassWord, ref System.Exception myException)
{
   try
   {
      DB2OdbcFileConnectionString myConnection = new DB2OdbcFileConnectionString(FileName,
false);
      myConnection.UserName = NameOfUser;
      myConnection.Password = PassWord;
      myConnection.Save();
      System.Exception MyEx= new System.Exception(@"Successful Creation", null);
      myException = MyEx;
      return myConnection;

   }
   catch (Exception ex)
   {
      myException = ex;
      return null;

   }
}
```

See Also

Tasks

How to Edit a Configuration

Reference

Configuring a Data Source for the ODBC Driver for DB2

Other Resources

Creating a Connection String

How to Retrieve Data

Creating connection string information requires that you create an object that is derived from the **IConnectionString** class, such as **DB2OdbcConnectionString** or **DB2OleDbConnectionString**. After you create the string, you can save, modify, or retrieve information from it by using the associated properties.

To retrieve and modify connection string information

1. Create a new connection string by calling the specific type of connection string constructor, using the file path of the .udl file that contains the specified connection string.

   Or, you can call **ReadUDL** for the specified **ConnectionString** type. Many of the **ConnectionString** classes also have a **Clone** method that you may want to use. Note that **Clone** does not load the current instance into active memory, but instead makes a copy that you can later modify and save to disk.

   If you are attempting to retrieve data from a connection string that you currently have an instance of, you can call **Load**. For example, if you recently created a new connection string and called **Save**, you can retrieve the object from storage and into active memory by calling **Load** on the object again.

   If you use a path that describes a file that does not exist, the system creates a .udl new file using the path described.

2. Retrieve the connection data from your current instance by using **GetString** or by accessing the relevant property.

   Using **GetString** enables you to manipulate the connection string as though it were a standard text string. In contrast, accessing the value as a property is usually simpler and safer.

3. When you are finished viewing or manipulating the relevant value, return the value to the object by calling **SetString** or by setting the appropriate property.

4. When you are finished, save your changes to secondary storage by calling **Save**.

The following code example demonstrates how to retrieve, change, and save connection string data.

```
static System.Exception ChangeCommentInUDL(string connString, string newComment)
{
   try
   {
      IConnectionString udl = DB2OleDbConnectionString.ReadUDL(connString);
      udl.Comment = newComment;
      udl.Save();
      System.Exception noException = null;
      return noException;

   }
   catch (System.Exception ex)
   {
      return ex;
   }
}
```

See Also

Tasks

How to Create a Connection String for an ODBC System, User or File DSN

How to Display an Initialization String

Other Resources

Creating a Connection String

Performing Administrative Tasks

The administrative tasks for the Data Access Library are the tasks you can perform using the **DataAccessControl** interface. In general, these tasks are those that require you go beyond creating universal data link (UDL) or host column description (HCD) files, and start interacting with the actual data sources.

In This Section

How to Convert Data Source Information

Converting a data source enables you to convert DB2 data source information from one provider type to another. Note that this does not change the data source itself, but rather reformats the data source information stored on your system.

To convert data source information

1. Retrieve the connection string for the relevant database using the appropriate call to **ReadUDL**.
2. Create a connection string to the converted database by calling **DataAccessControl.ConvertTo**, describing the data types of the original data and translated data.
3. Save the data in the new connection string by calling **IConnectionString.Save**.

The following example demonstrates how to convert data source information between OLE DB UDLs and ODBC DSNs.

```
IConnectionString ConvertUDLtoODBC(string originalUDL)
{
   try
   {
      IConnectionString udl = DB2OleDbConnectionString.ReadUDL(originalUDL);
      IConnectionString odbcDsn = DataAccessControl.ConvertTo(udl, typeof(DB2OdbcSysConnect
ionString));
      odbcDsn.Save();
      return odbcDsn;
   }
   catch
   {
      return null;
   }
}
```

See Also

Tasks

How to Convert Data Sources

Other Resources

Performing Administrative Tasks

How to Create a DB2 Package

A DB2 package is a collection of data used by a provider implemented as an IBM Distributed Relational Database Architecture (DRDA) application requester. The provider uses packages to issue SQL statements and call DB2 stored procedures. You can use CreatePackages to create these packages.

The create package command creates a Host Integration Server package on a DB2 system.

To create a DB2 package

1. Create a connection string to the targeted database with a call to ReadUDL.
2. Create the package with **CreatePackages**.

The following example describes how to create a package.

```
static void CreatePackage(string myUDL, System.Exception myException)
{
try
   {
      IConnectionString connString = DB2OleDbConnectionString.ReadUDL(myUDL);
      DataAccessControl.CreatePackages(connString, null);
      myException = null;
   }
catch (System.Exception Caught)
   {
      myException = Caught;
   }
}
```

See Also

Tasks

How to Create Packages

Reference

Creating Packages for Use with the OLE DB Provider for DB2

Concepts

Creating Packages for Use with the ODBC Driver for DB2

How to Test a Connection

After you create the appropriate connection string and store it in a file, you can programmatically test the validity of your connection with a call to TestConnection.

To test a connection

1. Create or retrieve the connection string on which you run the sample query.

   For more information about creating and retrieving connection strings, see Creating a Connection String and How to Retrieve Data.

2. Test the connection with a call to **TestConnection**.

   If successful, **TestConnection** returns the class and version number of the server. Otherwise, the method returns null. Note that you may be required to enter the user name and password. In this case, the **Password** dialog box appears.

See Also
Tasks
How to Test a Connection
Other Resources
Performing Administrative Tasks

How to Run a Sample Query

One of the final checks you can perform on a created connection string is a simple query that fetches a list of available files and data sources in the target database. Once you create the connection string, you can run this query using DataAccessControl.Microsoft.HostIntegration.DataAccessLibrary.DataAccessControl.SampleQuery(Microsoft.HostIntegration.DataAccessLibrary.IConnectionString).

To run a sample query

1. Create or retrieve the connection string on which you run the sample query.

   For more information on creating and retrieving connection strings, see Creating a Connection String and How to Retrieve Data.

2. Run the sample query with a call to **SampleQuery**.

   If successful, **SampleQuery** returns a data table with the relevant values. Otherwise, the method returns null.

See Also
Tasks
How to Run a Sample Query
Other Resources
Performing Administrative Tasks

Managed Provider Programmer's Guide

A Managed Provider is a .NET Framework data provider that is used for connecting Host Integration Server 2009 applications to a database, executing commands, and retrieving results. Those results are processed directly, placed in an ADO.NET DataSet to be exposed to the user, combined with data from multiple sources, or accessed remotely between tiers. A Host Integration Server Managed Provider is designed to be lightweight, creating a minimal layer between the data source and your code, and increasing performance without sacrificing functionality.

In This Section

Managed Provider for DB2 Programmer's Guide

This section describes the general architecture and implementation details of the Managed Provider for DB2. The Managed Provider for DB2 enables a client to connect to a DB2 data source in order to retrieve and modify data. The Managed Provider for DB2 implements a subset of the **System.Data** interfaces to enable connection management and query execution.

In This Section

Managed Provider for DB2

Host Integration Server 2009 includes a Managed Provider to access data from a loosely coupled data source: in this case, a DB2 database. The Managed Provider for DB2 uses two central .NET Framework components to implement this capability: the ADO.NET **DataSet** object and the Managed Provider for DB2 itself.

A **DataSet** object is designed for data access independent of any data source. As a result, it can be used with multiple and different data sources, with XML data, or to manage data local to the application. A **DataSet** contains a collection of one or more **DataTable** objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects.

In contrast, the Managed Provider for DB2 provides data manipulation and fast, forward-only, read-only access to data. Therefore, the Managed Provider for DB2 exposes a specific set of objects. The MSDb2Command object enables access to DB2 commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The MsDb2DataReader provides a high-performance stream of data from the DB2 database. Finally, the MsDb2DataAdapter provides the bridge between the **DataSet** object and the DB2 database. MsDb2DataAdapter uses MsDb2Command objects to execute SQL commands at the data source to both load the **DataSet** with data, and reconcile changes that were made to the data in the **DataSet** back to the data source.

In This Section

Managed Provider for DB2 Goals

The main goal of the Managed Provider for DB2 is to enable access to loosely coupled data sources using the technologies offered by ADO.NET and the .NET Framework. To implement this goal, the Managed Provider for DB2 was designed as an incremental improvement over the traditional COM-based OLE DB providers. In addition, the Managed Provider for DB2 is optimized for DB2 over the currently available OLE DB .NET Framework data provider.

ODBC and OLE DB Solutions

Microsoft has offered programming models and tools for developing enterprise data integration solutions for several years. These include the industry-standard ODBC and the COM-based OLE DB. However, developing and deploying ADO-based Web solutions presents a number of issues, including insufficient design tools, decreased performance, limited scalability, and little XML interoperability.

Additionally, the ADO- and OLE DB-based data architecture is based on solutions that require tightly coupled, live connections between application and data tiers. However, the vast majority of Web solutions today require a loosely coupled association between application components and tiers. Also, most modern Web application requirements include the use of XML as the universal Web message and data medium. Remote Data Services (RDS) offered a disconnected recordset between application and presentation tiers. ADO also offered the ability to persist recordsets as XML. However, there was no uniform way to provide asynchronous execution between tiers. In addition, XML recordset persistence was extremely limited.

ADO.NET and the .NET Framework

To make implementation easier and to improve performance, scalability, and XML support, Microsoft offered the .NET Framework and ADO.NET. The .NET Framework offered developers a number of advantages over COM, such as cross-language inheritance, object lifetime management, and multilanguage class libraries as supported by the common language runtime (CLR). The CLR offers developers a predictable, managed environment in which to execute their programs. The ADO.NET programming model was designed to provide better performance and scalability than the older ADO. ADO.NET is a set of common classes for exposing data services that are implemented by .NET Framework data providers.

Microsoft Visual Studio has shipped two .NET Framework data provider implementations: the first provider accesses Microsoft SQL Server through an application-level protocol called Tabular Data Stream (TDS), and is called the SQL Server .NET Framework Data Provider. The second provider accesses the underlying OLE DB providers, and is called the OLE DB .NET Framework Data Provider.

However, just as the OLE DB Provider for ODBC offers less optimal performance and scalability than a direct OLE DB provider solution, so too does the OLE DB .NET Framework Data Provider offer a less optimal solution than a specific .NET Framework data provider implementation would. In addition, the OLE DB .NET Framework Data Provider adds complexity and compatibility issues with an extra layer of indirection. Therefore, Host Integration Server has implemented a .NET Framework data provider for a key enterprise data source: DB2.

See Also

Other Resources

Managed Provider for DB2 Programmer's Guide

Relationships between the .NET Provider for DB2 Interfaces

The Managed Provider for DB2 interfaces interact in different ways—with the exception of MsDb2DataAdapter, the remaining classes adhere to a rigid parent/child relationship:

- An MsDb2Connection can have one MsDb2Transaction running, although it can have multiple MsDb2Commands.
- An MsDb2Transaction can have one or more MsDb2Commands running.
- An MsDb2Command owns one MsDb2ParameterCollection, which stores multiple MsDb2Parameters.

  An MsDb2Command can also create a single MsDb2DataReader for parsing one or more resultsets.

The MsDb2DataAdapter takes advantage of all the other Managed Provider interfaces. The MsDb2DataAdapter serves as the gateway between a host DB2 system and a client-side ADO.NET DataSet. The DataSet is an important piece of the .NET data framework because it provides a mechanism for caching data in a managed environment and inferring XML schema information, basically providing a gateway between DB2 data and Microsoft Web services.

See Also
Concepts
Examining the Core Interface for a Managed Provider
Other Resources
Managed Provider for DB2

Using the Managed Provider for DB2

The Managed Provider for DB2 operates in most ways as a normal data provider: you can connect to a DB2 database, execute commands, retrieve data, and use stored procedures.

In This Section

Using the Managed Provider for DB2 with Visual Studio

You can create a project that uses the managed provider for DB2 with Visual Studio as you would any other project. The following topics describe any issues that are unique to this provider.

Data Design Tools

The Make Table option in the Visual Studio 2005 Query Designer is not supported for the Managed Provider for DB2 in conjunction with IBM DB2 databases. Therefore, the syntax generated by the SELECT INTO statement in Visual Studio is not supported by DB2 when executed using dynamic SQL. There is no solution for this problem.

See Also

Other Resources

Using the Managed Provider for DB2

Connecting to and Disconnecting from a DB2 Database

There are two ways to connect and disconnect to a DB2 database. The first and most common way is using an MsDb2Connection to open and close the connection. The second method is to access a connection pool.

In This Section

How to Connect with an MsDb2Connection

The first step in accessing a remote DB2 database is to connect to the database. You must use MsDb2Connection to access an IBM DB2 data source. After you have connected, you can retrieve, modify, and update any information that you want. Note that connections are not implicitly released when the MsDb2Connection falls out of scope or is reclaimed by garbage collection. Therefore, you must close the connection when you are finished using it. You can close a connection by using either **MsDb2Connection.Close** or **MsDb2Connection.Dispose**.

Example

The following example demonstrates how to connect to a DB2 database.

```
Public void SampleConnect()
{
   MsDb2Connection myConnection = null;
   Try
   {
      myConnection = new MsDb2Connection(@"file name=HOST.udl ");
      myConnection.Open();
      // Perform any necessary tasks here.
      myConnection.Close();
   }
finally
   {
      if(myConnection!= null)
         myConnection.Dispose();
   }
}
```

See Also

Tasks

How to Connect to and Disconnect from a Host File System

Other Resources

Using the Managed Provider for DB2

How to Connect to a DB2 Connection Pool

Another way to connect to a DB2 database is through a *connection pool*. Although implemented differently on the server, a connection pool is identical to a traditional connection from the perspective of a client application.

A connection pool is a set of one or more connections that the server keeps open to service requests from one or more clients. When the client is finished with a connection, the server does not terminate the connection. Instead, the connection is released back into the pool, and can then service another client. Connection pooling is frequently used in situations where clients connect, query, and terminate a connection to the server multiple times over the course of a session, such as a database server that is accessed through the Internet.

To connect and disconnect to a DB2 connection pool Using Host Integration Server

1. Connect to the DB2 server with MsDb2Connection, with the Microsoft.HostIntegration.MsDb2Client.MsDb2Connection.ConnectionPooling set to **true**.
2. Perform your queries as you would with a traditional DB2 connection.
3. Use **MsDb2Connection.Close** or **MsDb2Connection.Dispose** to end your session.

Note
Calling **Close** or **Dispose** on a connection from a connection pool does not actually close or dispose the connection. In stead, the server returns the connection to the pool.

The following code example shows how to connect to a DB2 database using a connection pool.

```
int GetNumberOfOrders()
{
    MsDb2Connection conn = new MsDb2Connection(@"File Name=C:\MyConn.UDL");
    sDb2Command cmd;
    int numOrders = 0;
    conn.ConnectionPooling = true;
    conn.Open();
    cmd = new MsDb2Command("select count(*) from orders", conn);
    numOrders = (int)cmd.ExecuteScalar();
    conn.Close();
    return numOrders;
}
```

See Also

Other Resources

Using the Managed Provider for DB2

Connecting to and Disconnecting from a DB2 Database

Working with Connection Strings and the Managed Provider for DB2

A connection string contains initialization information passed as a parameter from a data provider to a data source, which is then parsed immediately after being set. Syntax errors generate a run-time exception, but other errors can be found only after the data source has validated the information in the connection string. After the information is validated, the data source sets various connection string options that enable the connection and allow it to be opened.

The .NET Framework 2.0 provides new capabilities for working with connection strings, such as the **MsDb2ConnectionStringBuilder** class, which facilitates building valid connection strings at run time based on user input.

In This Section

Building Connection Strings

The Managed Provider for DB2 provides a strongly typed connection string builder class that inherits from **DbConnectionStringBuilder**. The connection string builders let developers programmatically create syntactically correct connection strings that are based on user input, and also parse and rebuild existing connection strings by using methods of the class. The MsDb2ConnectionStringBuilder class provides strongly typed properties that correspond to the known key/values pairs allowed by the Managed Provider for DB2.

In earlier versions of ADO.NET, there was no compile-time checking of connection strings that consisted of concatenated string values. At run time, an incorrect keyword would generate an invalid ArgumentException. Because values received from a user were not checked or quoted appropriately, it was possible for an attacker to bypass expected settings.

The **MsDb2ConnectionStringBuilder** class performs checks for valid key/value pairs. An invalid pair throws an exception and injected values are handled in a safe manner. Each class maintains a fixed collection of synonyms and can translate from a synonym to the corresponding well-known key name.

Example
See Also
Other Resources
Working with Connection Strings and the Managed Provider for DB2

Using Connection String Keywords

The format of a connection string is a semicolon-delimited list of key/value parameter pairs:

keyword1=value; keyword2=value

When the connection string is validated by the data source, spaces are ignored and keywords are not case sensitive. However, values may be case sensitive, depending on the case sensitivity of the data source. To include values that contain a semicolon, single-quote character, or double-quote character, the value must be enclosed in double quotation marks.

To avoid misspellings, the Managed Provider for DB2 exposes the keywords and values of the connection string as properties.

Example
See Also
Other Resources
Working with Connection Strings and the Managed Provider for DB2

Storing and Retrieving Connection Strings

We recommend that you not embed connection strings in your code. If the location of the server ever changes, your application must be recompiled. In addition, unencrypted connection strings that are compiled into an application's source code can be viewed using the MSIL Disassembler (ildasm.exe).

To avoid storing connection strings in your code, you can store them in the web.config file for an ASP.NET application and in the app.config file for a Windows application. Although configuration files are mainly used in ASP.NET applications, where they are protected from being browsed to, you can also use them in a Windows application. The syntax and format of the app.config and web.config files is identical for both ASP.NET and Windows applications.

The connection string can be stored in the configuration file in the **<connectionStrings>** element. Connection strings are stored as key/value pairs, where the name key can be used to look up the value stored in the **connectionString** attribute at run time.

You can work with configuration files programmatically by using the **MsDb2Configuration** class. The methods of the **WebConfigurationManager** object also enable you to obtain a configuration section, each of which has its own object type. In addition, the **System.Configuration** namespace provides classes for working with configuration information stored in configuration files.

Example
See Also
Other Resources
Working with Connection Strings and the Managed Provider for DB2

Executing Commands in a DB2 Database

The MsDb2Command object exposes several **Execute** methods that you can use to perform the intended action. When you are returning results as a stream of data, use ExecuteReader to return a **DataReader** object. Use ExecuteScalar to return a singleton value. Use ExecuteNonQuery to execute commands that do not return rows.

Using MsDb2Command with Stored Procedures

When you use the **MsDb2Command** object with a stored procedure, you can set the **CommandType** property of the **MsDb2Command** object to **StoredProcedure**. With a **CommandType** of **StoredProcedure**, you can use the **Parameters** property of the **Command** to access input and output parameters and return values. The **Parameters** property can be accessed regardless of the **Execute** method called. However, when you call **ExecuteReader**, return values and output parameters cannot be accessed until the **DataReader** is closed.

Note that SQL statements that modify data (such as **INSERT**, **UPDATE**, or **DELETE**) do not return rows. Similarly, many stored procedures perform an action but do not return rows. To execute commands that do not return rows, create an **MsDb2Command** object with the appropriate SQL command and an MsDb2Connection, including any required MsDb2Parameter**s**. Execute the command by using the **ExecuteNonQuery** method of the **MsDb2Command** object. The **ExecuteNonQuery** method returns an integer that represents the number of rows affected by the statement or stored procedure that was executed. If multiple statements are executed, the value returned is the sum of the records affected by all the statements executed.

Modifying Databases and Catalogs

To execute a command to modify a database or catalog, such as the **CREATE TABLE** or **CREATE PROCEDURE** statement, create an **MsDb2Command** object by using the appropriate SQL statements and an **MsDb2Connection** object. Execute the command by using the **ExecuteNonQuery** method of the **MsDb2Command** object.

See Also

Other Resources

Using the Managed Provider for DB2
Managed Provider for DB2 Programmer's Guide
Managed Provider for DB2 Programmer's Reference

Reading Data from a DB2 Database

In addition to executing commands on a remote database, you can also retrieve information from the database to view locally.

In This Section

Reading Data from a Database

You can use MsDb2DataReader to retrieve a read-only, forward-only stream of data from a database. Using **MsDb2DataReader** can increase application performance and reduce system overhead because only one row at a time is ever in memory.

After you create an instance of the MsDb2Command object, you can create an **MsDb2DataReader** by calling ExecuteReader to retrieve rows from a data source.

You can use Microsoft.HostIntegration.MsDb2Client.MsDb2DataReader.Read to obtain a row from the results of the query. You access each column of the returned row by passing the name or ordinal reference of the column to **MsDb2DataReader**. However, for best performance, **MsDb2DataReader** provides a series of methods that enable you to access column values in their native data types. Using the typed accessor methods when the underlying data type is known reduces the amount of type conversion required when retrieving the column value.

MsDb2DataReader also provides a nonbuffered stream of data that enables procedural logic to efficiently process results from a data source sequentially. **MsDb2DataReader** is a good choice when you are retrieving large amounts of data because the data is not cached in memory.

After you are finished with **MsDb2DataReader**, be sure to call the **Close** method. In addition, output parameters and return values from an **MsDb2Command** are not available until **MsDb2DataReader** is closed.

Note

The Distributed Relational Data Architecture (DRDA) uses a "." as a decimal point and a "," to separate numerical values. If yo u are working in a language, such as German, that uses a "," as a decimal point, you might receive an error when you are retri eving data from your database. To avoid this error, use **System.Globalization.CultureInfo.InvariantCulture** when you call the **ToString** and **Parse** methods.

Example

The following example shows how to read data from a DB2 database by reading a row from a **SELECT** statement:

```
Public void ReadMyData(string myConnString)
{
   string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
   MsDb2Connection myConnection = new MsDb2Connection(myConnString);
   MsDb2Command myCommand = new MsDb2Command(mySelectQuery,myConnection);
   myConnection.Open();
   MsDb2DataReader myReader;
   myReader = myCommand.ExecuteReader();
   // Always call Read before accessing data.
   While (myReader.Read())
   {
       Console.WriteLine(myReader.GetInt32(0) + ", "
       + myReader.GetString(1));
   }
   // Always close when done reading.
   myReader.Close();
   // Close the connection when done.
   myConnection.Close();
}
```

See Also

Other Resources

Using the Managed Provider for DB2
Managed Provider for DB2 Programmer's Guide
Managed Provider for DB2 Programmer's Reference

How to Retrieve Multiple Result Sets

MsDb2DataReader provides the NextResult method to iterate through multiple returned results.

Example

The following example shows how to retrieve multiple result sets.

```
Public void ReadMyData(string myConnString)
{
   string myCallQuery = "CALL MyReports()";
   MsDb2Connection myConnection = new MsDb2Connection(myConnString);
   MsDb2Command myCommand = new MsDb2Command(myCallQuery,myConnection);
   myConnection.Open();
   MsDb2DataReader myReader;
   myReader = myCommand.ExecuteReader();
   do
      {
      // Always call Read before accessing data.
      While (myReader.Read())
         {
         Console.WriteLine(myReader.GetString(0) + " , ", "
                 + myReader.GetString(1));
         }
       }
   while(myReader.NextResult());
   // Always call Close when done reading.
   myReader.Close();
   // Close the connection when done with it.
   myConenction.Close();
}
```

See Also

Other Resources

Using the Managed Provider for DB2

Managed Provider for DB2 Programmer's Guide

Managed Provider for DB2 Programmer's Reference

Retrieving Schema Information

While an MsDb2DataReader is open, you can retrieve schema information about the current result set by using **MsDb2DataReader.**GetSchemaTable. **GetSchemaTable** returns a data table that is populated with rows and columns that contain the schema information for the current result set. The data table contains one row for each column of the result set. Each column of the schema table row maps to a property of the column returned in the result set, where the **ColumnName** is the name of the property and the value of the column is the value of the property.

See Also

Other Resources

Using the Managed Provider for DB2
Managed Provider for DB2 Programmer's Guide
Managed Provider for DB2 Programmer's Reference

Using Stored Procedures in a DB2 Database

Stored procedures offer many advantages in data-driven applications. By using stored procedures, you can encapsulate database operations in a single command, optimized for best performance, and enhanced with additional security. Although you can call a stored procedure by passing the stored procedure name followed by parameter arguments as an SQL statement, using the Parameters collection of MsDb2Command object enables you to more explicitly define stored procedure parameters, and also to access output parameters and return values.

To call a stored procedure, set the CommandType of the **MsDb2Command** object to **StoredProcedure**. After the **CommandType** is set to **StoredProcedure**, you can use the **Parameters** collection to define parameters.

You can create an MsDb2Parameter object by using the **MsDb2Parameter** constructor, or by calling the **Add** method of the **Parameters** collection of an **MsDb2Command**. **MsDb2Parameters.Add** takes as input either constructor arguments or an existing **MsDb2Parameter object**. When setting the Value of an **MsDb2Parameter** to a null reference, use **DBNull.Value**.

For parameters other than Input parameters, you must set the **ParameterDirection** property to specify whether the parameter type is **InputOutput**, **Output**, or **ReturnValue**.

See Also

Other Resources

Using the Managed Provider for DB2

How to Obtain a Single Value from a DB2 Database

You might need to return database information that is just a single value rather than in the form of a table or data stream. For example, you might want to return the result of an aggregate function such as Count(*), Sum(Price), or Avg(Quantity). The **Command** object enables you to return single values by using the **ExecuteScalar** method. The **ExecuteScalar** method returns as a scalar value the value of the first column of the first row of the result set.

Example

The following code example demonstrates how to retrieve a single value from a DB2 database.

```
static void SingleValueConnection()
{
    MsDb2Connection myConnection = null;
    try
    {
// Obtaining a single value as a database.
        myConnection = new MsDb2Connection(@"file name=HOST.udl ");
        myConnection.Open();
        MsDb2Command myCommand = new MsDb2Command("SELECT Count(*) FROM Orders", myConnecti
on);
       Int32 count = (Int32)myCommand.ExecuteScalar();
        myConnection.Close();
    }
        catch (Exception exc)
    {
        Console.WriteLine(exc);
        Console.ReadLine();
    }

    finally
    {
        if(myConnection!= null)
            myConnection.Dispose();
    }
} // End SingleValueConnection.
```

See Also

Other Resources

Using the Managed Provider for DB2

Working with the DataAdapter and the DataSet for a DB2 Database

The MsDb2DataAdapter is the main interface for retrieving and updating the **DataSet**.

In This Section

Populating a Managed Provider Dataset from a Data Adapter

The dataset is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The dataset represents a complete set of data including tables, constraints, and relationships among the tables. Because the dataset is independent of the data source, a dataset can include data that is local to the application, and also data from multiple data sources. Interaction with existing data sources is controlled through an **MsDb2DataAdapter** object.

The Microsoft.HostIntegration.MsDb2Client.MsDb2DataAdapter.SelectCommand is a **Command** object that retrieves data from the data source. The **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties of MsDb2DataAdapter are also **Command** objects that manage updates to the data in the data source according to modifications made to the data in the dataset.

Fill Method

The **MsDb2DataAdapter.Fill** method is used to populate a dataset with the results of **Microsoft.HostIntegration.MsDb2Client.MsDb2DataAdapter.SelectCommand**. **Fill** takes as its arguments a **DataSet** object to be populated, and a **DataTable** object, or the name of the **DataTable** object to be filled with the rows returned from **SelectCommand**.

The **Fill** method uses MsDb2DataReader implicitly to return the column names and types that are used to create the tables in the dataset, and also the data to populate the rows of the tables in the dataset. Tables and columns are only created if they do not already exist; otherwise **Fill** uses the existing dataset schema. Column types are created as .NET Framework types. Primary keys are not created unless they are located in the data source, and **DataAdapter.MissingSchemaAction** is set to **MissingSchemaAction.AddWithKey**. If **Fill** finds that a primary key exists for a table, it overwrites data in the **DataSet** with data from the data source for rows where the primary key column values match those of the row returned from the data source. If no primary key is found, the data is appended to the tables in the **DataSet. Fill** uses any mappings that may exist when populating the **DataSet** object.

Note
If **SelectCommand** returns the results of an OUTER JOIN, the **MsDb2DataAdapter** does not set a PrimaryKey value for the resulting **DataTable**. You must define the PrimaryKey yourself to ensure that duplicate rows are resolved correctly

Multiple Result Sets

If **MsDb2DataAdapter** encounters multiple result sets, it creates multiple tables in the dataset. The tables are given an incremental default name of TableN, starting with "Table" for Table0. If a table name is passed as an argument to the **Fill** method, the tables are given an incremental default name of TableNameN, starting with "TableName" for TableName0.

Any number of **MsDb2DataAdapter** objects can be used with a dataset. Each **MsDb2DataAdapter** can be used to fill one or more **DataTable** objects and resolve updates back to the relevant data source. You can add **DataRelation** and **Constraint** objects to the dataset locally, enabling you to relate data from dissimilar data sources.

See Also

Other Resources

Working with the DataAdapter and the DataSet for a DB2 Database

Using the Managed Provider for DB2

Working with DataAdapter Events

MsDb2DataAdapter exposes two events you can use to respond to changes made to data at the data source. The following table shows the **MsDb2DataAdapter** events.

Event	Description
RowUpdating	An UPDATE, INSERT, or DELETE operation on a row (by a call to one of the **Update** methods) is about to begin.
RowUpdated	An UPDATE, INSERT, or DELETE operation on a row (by a call to one of the **Update** methods) is complete.

RowUpdating is raised before any update to a row from the dataset has been processed at the data source. **RowUpdated** is raised after any update to a row from the dataset has been processed at the data source. As a result, you can use **RowUpdating** to modify update behavior before it occurs, to provide additional handling when an update occurs, to retain a reference to an updated row, to cancel the current update and schedule it for a batch process to be processed later, and so on. **RowUpdated** is useful for responding to errors and exceptions that occur during the update. You can add error information, retry logic, and so on, to the dataset.

Arguments

The *MsDb2RowUpdatingEventArgs* and *MsDb2RowUpdatedEventArgs* arguments that are passed to the **RowUpdating** and **RowUpdated** events include the following:

- A **Command** property that references the **Command** object that is used to perform the update.
- A **Row** property that references the **DataRow** object containing the updated information.
- A **StatementType** property for what type of update is being performed.
- The **TableMapping**, if applicable.
- The **Status** of the operation.

You can use the **Status** property to determine whether an error has occurred during the operation and, if you want, to control the actions against the current and resulting rows. When the event occurs, the **Status** property equals either **Continue** or **ErrorsOccurred**.

Status Property Values

The following table shows the values to which you can set the **Status** property in order to control subsequent actions during the update.

Status	Description
Continue	Continue the update operation.
ErrorsOccurred	Abort the update operation and throw an exception.
SkipCurrentRow	Ignore the current row and continue the update operation.
SkipAllRemainingRows	Abort the update operation, but do not throw an exception.

Setting the **Status** property to **ErrorsOccurred** causes an exception to be thrown. You can control which exception is thrown by setting the **Errors** property to the desired exception. Using one of the other values for **Status** prevents an exception from being thrown.

See Also

Other Resources

Working with the DataAdapter and the DataSet for a DB2 Database

Using the Managed Provider for DB2

Updating the DB2 Database with a Data Adapter and the Dataset

The **Update** method of **MsDb2DataAdapter** is called to resolve changes from a dataset back to the data source. The **Update** method, like the **Fill** method, takes as arguments an instance of **DataSet**, and an optional **DataTable** object or **DataTable** name. The **DataSet** instance is the **DataSet** object that contains the changes that have been made, and the **DataTable** object identifies the table from which to retrieve the changes.

Calling the Update Method

When you call the **Update** method, **MsDb2DataAdapter** analyzes the changes that have been made and executes the appropriate command (INSERT, UPDATE, or DELETE). When the **MsDb2DataAdapter** encounters a change to a data row, it uses **InsertCommand**, **UpdateCommand**, or **DeleteCommand** to process the change. This enables you to maximize the performance of your ADO.NET application by specifying command syntax at design time and, where possible, with stored procedures. You must explicitly set the commands before calling **Update**. If **Update** is called, and the appropriate command does not exist for a particular update (for example, no **DeleteCommand** for deleted rows), an exception is thrown. You can use command parameters to specify input and output values for an SQL statement or stored procedure for each modified row in a dataset.

If your **DataTable** object maps to or is generated from a single database table, you can take advantage of the **MsDb2CommandBuilder** object to automatically generate the **DeleteCommand**, **InsertCommand**, and **UpdateCommand** properties of the **MsDb2DataAdapter** object.

Updating Datasets

The **Update** method resolves your changes back to the data source; however other clients might have modified data at the data source since the last time you filled the dataset. To update your dataset with current data, use the **MsDb2DataAdapter** and **Fill** method. New rows are added to the table, and updated information is incorporated into existing rows. The **Fill** method determines whether a new row will be added or an existing row will be updated by examining the primary key values of the rows in the dataset and the rows returned by **SelectCommand**. If the **Fill** method encounters a primary key value for a row in the dataset that matches a primary key value from a row in the results returned by **SelectCommand**, it updates the existing row with the information from the row returned by **SelectCommand** and sets the **RowState** property of the existing row to **Unchanged**. If a row returned by **SelectCommand** has a primary key value that does not match any of the primary key values of the rows in the dataset, the **Fill** method adds a new row with a **RowState** of **Unchanged**.

Note
If **SelectCommand** returns the results of an OUTER JOIN, the **DataAdapter** object does not set a PrimaryKey value for the r esulting **DataTable** object. You must define the PrimaryKey yourself to ensure that duplicate rows are resolved correctly.

To handle exceptions that might occur when you call the **Update** method, you can use the **RowUpdated** event to respond to row update errors as they occur, or you can set **DataAdapter.ContinueUpdateOnError** to **true** before calling **Update**, and respond to the error information stored in the **RowError** property of a particular row when the update is complete.

Note
Calling **AcceptChanges** on the **DataSet**, **DataTable**, or **DataRow** objects causes all Original values for a data row to be ove rwritten with the Current values for the data row. If the field values that identify the row as unique have been modified, after you call **AcceptChanges**, the Original values no longer match the values in the data source.

Working with Auto-Incrementing Columns

If the tables from your data source have auto-incrementing columns, you can fill the columns in your dataset either by returning the auto-increment value as an output parameter of a stored procedure and mapping that to a column in a table, or by using the **RowUpdated** event of the **MsDb2DataAdapter**.

However, the values in your dataset can become out of sync with the values at the data source and result in unexpected behavior. For example, consider a table that has an auto-incrementing primary key column of CustomerID. If you add two new customers within the dataset, they receive auto-incremented CustomerId values of 1 and 2. When the second customer row is passed to the **Update** method of **MsDb2DataAdapter**, the newly added row receives an auto-incremented CustomerID value of 1 at the data source, which does not match the value, 2, in the dataset. When the **MsDb2DataAdapter** fills the row in the dataset with the returned value, a constraint violation occurs because the first customer row already has a CustomerID of 1.

To avoid this behavior, we recommend that, when you are working with auto-incrementing columns at a data source and auto-incrementing columns in a dataset, you create the column in the dataset with an AutoIncrementStep of -1 and an AutoIncrementSeed of 0, and also ensure that your data source generates auto-incrementing Identity values starting at 1 and incrementing with a positive step value. As a result, the dataset will generate negative numbers for auto-incremented values that do not conflict with the positive auto-increment values generated by the data source. Another option is to use columns of type **Guid** instead of auto-incrementing columns. The algorithm that generates **Guid** values should never generate the same **Guid** in the dataset as is generated by the data source.

In many circumstances, the order in which changes that are made through the dataset are sent to the data source is important. For example, if a primary key value for an existing row is updated, and a new row has been added with the new primary key value, it is important to process the update before the insert.

You can use the **Select** method of the **DataTable** object to return a **DataRow** array that only references rows with a particular **RowState**. You can then pass the returned **DataRow** array to the **Update** method of the **MsDb2DataAdapter** to process the modified rows. By specifying a subset of rows to be updated, you can control the order in which inserts, updates, and deletes are processed.

See Also

Other Resources

Working with the DataAdapter and the DataSet for a DB2 Database

Using Parameters with the DB2 DataAdapter

The **DataAdapter** class has four properties that are used to retrieve data from and update data to the data source: the **SelectCommand** property returns data from the data source; the **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties are used to manage changes at the data source. You must set the **SelectCommand** property before calling the **Fill** method of the **DataAdapter** object. The **InsertCommand**, **UpdateCommand**, or **DeleteCommand** properties must be set before the **Update** method of the **DataAdapter** is called, depending on what changes were made to the data in the **DataSet**. For example, if rows have been added, you must set the **InsertCommand** property before calling the **Update** method. When **Update** is processing an inserted, updated, or deleted row, the **DataAdapter** object uses the respective **Command** property to process the action. Current information about the modified row is passed to the **Command** object through the **Parameters** collection.

When you are updating a row at the data source, you call the UPDATE statement, which uses a unique identifier to identify the row in the table to be updated. The unique identifier is commonly the value of a primary key field. The UPDATE statement uses parameters that contain both the unique identifier, and the columns and values to be updated.

Specifying the **Parameter** type converts the value of the Parameter to the Managed Provider for DB2 before passing the value to the data source. You may also specify the type of a **Parameter** in a generic fashion by setting the **DbType** property of the **Parameter** object to a particular **DbType**.

ParameterDirection Enumeration Values

The following table shows the values you can use with the **ParameterDirection** enumeration to set the **Direction** of the **Parameter**.

Member name	Description
Input	The parameter is an input parameter. This is the default.
InputOutput	The parameter is capable of both input and output.
Output	The parameter is an output parameter.
ReturnValue	The parameter represents a return value.

SourceColumn and **SourceVersion** may be passed as arguments to the **Parameter** constructor, or set as properties of an existing **Parameter**. The **SourceColumn** is the name of the **DataColumn** from the **DataRow** where the value of the **Parameter** is retrieved. The **SourceVersion** specifies which **DataRow** version the **DataAdapter** uses to retrieve the value.

DataRowVersion Enumeration Values

The following table shows the **DataRowVersion** enumeration values available for use with **SourceVersion**.

Member name	Description
Current	The parameter uses the current value of the column. This is the default.
Default	The parameter uses the DefaultValue of the column.
Original	The parameter uses the original value of the column.
Proposed	The parameter uses a proposed value.

You can control how the values returned from the data source are mapped back to the **DataSet** object by using the **UpdatedRowSource** property of the **Command** object. By setting the **UpdatedRowSource** property to one of the **UpdateRowSource** enumeration values, you can control whether parameters returned by the **DataAdapter** command are ignored or applied to the changed row in the **DataSet** object. You can also specify whether the first returned row (if it exists) is applied to the changed row in the **DataSet** object.

UpdateRowSource Enumeration Values

The following table describes the different values of the **UpdateRowSource** enumeration and how they affect the behavior of a command used with a **DataAdapter** object.

Member name	Description
Both	Both the output parameters and the first row of a returned result set can be mapped to the changed row in the **DataSet**.
ReturnedFirstRecord	Only the data in the first row of a returned result set can be mapped to the changed row in the **DataSet**.
None	Any output parameters or rows of a returned result set are ignored.
OutputParameters	Only output parameters may be mapped to the changed row in the **DataSet**.

See Also

Other Resources

Working with the DataAdapter and the DataSet for a DB2 Database

Adding Constraints to a DB2 DataSet

The **Fill** method of the **MsDb2DataAdapter** fills a dataset only with table columns and rows from a data source; though constraints are commonly set by the data source, the **Fill** method does not add this schema information to the dataset by default. To populate a dataset with existing primary key constraint information from a data source, you can either call the **FillSchema** method of the **MsDb2DataAdapter**, or set the **MissingSchemaAction** property of the **MsDb2DataAdapter** to **AddWithKey** before calling **Fill**. This ensures that primary key constraints in the dataset reflect those at the data source. Foreign key constraint information is not included and must be created explicitly.

Adding schema information to a **DataSet** object before filling it with data ensures that primary key constraints are included with the **DataTable** objects in the **DataSet** object. As a result, when additional calls to fill the **DataSet** are made, the primary key column information is used to match new rows from the data source with current rows in each **DataTable** object, and current data in the tables is overwritten with data from the data source. Without the schema information, the new rows from the data source are appended to the **DataSet**, resulting in duplicate rows.

Note

If a column in a data source is identified as auto-incrementing, the **FillSchema** method, or the **Fill** method with a **MissingSchemaAction** of **AddWithKey**, creates a **DataColumn** with an **AutoIncrement** property set to **true**. However, you must set the **AutoIncrementStep** and **AutoIncrementSeed** values yourself.

Using the **FillSchema** method or setting the **MissingSchemaAction** to **AddWithKey** requires extra processing at the data source to determine primary key column information. This additional processing can hinder performance. If you know the primary key information at design time, we recommend that you explicitly specify the primary key column or columns in order to achieve optimal performance.

If the **MsDb2DataAdapter** encounters multiple result sets returned from the **SelectCommand** property, it creates multiple tables in the dataset. The tables are given a zero-based incremental default name of TableN, starting with Table instead of "Table0". If a table name is passed as an argument to the **FillSchema** method, the tables are given a zero-based incremental name of TableNameN, starting with TableName instead of "TableName0".

Setting up DataTable and DataColumn Mappings for a DB2 Database

An **MsDb2DataAdapter** contains a collection of zero or more **DataTableMapping** objects in its **TableMappings** property. A **DataTableMapping** object provides a master mapping between the data returned from a query against a data source, and a **DataTable** object. The **DataTableMapping** name can be passed instead of the **DataTable** name to the **Fill** method of the **MsDb2DataAdapter**.

A **DataTableMapping** object enables you to use column names in a **DataTable** object that are different from those in the database. The **MsDb2DataAdapter** uses the mapping to match the columns when the table is updated.

If you do not specify a **TableName** or a **DataTableMapping** name when calling the **Fill** or **Update** method of the **MsDb2DataAdapter**, the **MsDb2DataAdapter** looks for a **DataTableMapping** named "Table". If that **DataTableMapping** does not exist, the **TableName** of the **DataTable** object is "Table". You can specify a default **DataTableMapping** by creating a **DataTableMapping** with the name of "Table".

When the **Fill** method is passed an instance of a **DataSet** and a **DataTableMapping** name, and if a mapping with that name exists, it is used; otherwise, a **DataTable** object with that name is used.

Note

If a source column name is not supplied for a column mapping, or a source table name is not supplied for a table mapping, d efault names are automatically generated. If no source column is supplied for a column mapping, the column mapping is giv en an incremental default name of SourceColumnN, starting with SourceColumn1. If no source table name is supplied for a t able mapping, the table mapping is given an incremental default name of SourceTableN, starting with SourceTable1.

Note

We recommend that you avoid the naming convention of SourceColumnN for a column mapping, or SourceTableN for a tabl e mapping, because the name you supply may conflict with an existing default column mapping name in the **ColumnMappi ngCollection** or table mapping name in the **DataTableMappingCollection**. If the supplied name already exists, an excepti on is thrown.

If **SelectCommand** returns multiple tables, **Fill** automatically generates table names with incremental values for the tables in the dataset, starting with the specified table name and continuing on in the form TableNameN, starting with TableName1. You can use table mappings to map the automatically generated table name to a name you want specified for the table in the dataset.

See Also

Other Resources

Working with the DataAdapter and the DataSet for a DB2 Database

Using the Managed Provider for DB2

How to Perform Transactions with a DB2 Database

Transactions are a group of database operations combined into a logical unit of work, and are used to control and maintain the consistency and integrity of each database despite errors that might occur in the system. A transaction consists of a series of SQL SELECT, INSERT, UPDATE, or DELETE statements. If no errors occur during a transaction, all modifications in the transaction become a permanent part of the database. If errors occur, none of the modifications are made to the database.

A transaction is considered to be a local transaction when it is a single-phase transaction and is handled by the database directly. Transactions are considered to be distributed transactions when they are coordinated by a transaction monitor and use fail-safe mechanisms (such as two-phase commit) for transaction resolution.

Note
Transactions are most efficient when they are performed on the server. If you are working with a SQL Server database that m akes extensive use of explicit transactions, you should consider writing them as stored procedures using the Transact-SQL BE GIN TRANSACTION statement.

You control transactions with the **MsDb2Connection** object. You can initiate a local transaction with the **BeginTransaction** method. Once you have begun a transaction, you can enlist a command in that transaction with the **Transaction** property of an **MsDb2Command** object. You can then commit or roll back modifications made at the data source based on the success or failure of the components of the transaction.

There are three basic commands for transactions: BEGIN, COMMIT, and ROLLBACK. The BEGIN statement marks the beginning of a transaction. All procedures attempted after the BEGIN statement are considered part of the transaction, which is completed by the COMMIT statement, or canceled by the ROLLBACK statement.

To perform a transaction

1. Call **MsDb2Connection.BeginTransaction** to mark the start of the transaction.

   BeginTransaction returns a reference to the transaction. Retain this reference so that you can assign it to commands that are enlisted in the transaction.

2. Assign the transaction to the **MsDb2Command.Transaction** to be executed.

   If a command is executed on a connection with an active transaction and the **Transaction** object has not been assigned to the **Transaction** property of the **Command**, an **MsDb2Exception** is thrown.

3. Execute the required commands.

4. Call **MsDb2Transaction.Commit** to complete the transaction, or call **MsDb2Transaction.Rollback** to cancel the transaction.

   If the connection is closed or disposed before either the **Commit** or **Rollback** methods have been executed, the transaction is rolled back.

The following code example demonstrates how to perform a transaction.

```
static void TransactionConnection()
{
    MsDb2Connection myConnection new MsDb2Connection(@"file name=HOST.udl ");
    myConnection.Open();
    // Start a local transaction.
    MsDb2Transaction myTrans = myConnection.BeginTransaction();
    // Enlist the command in the current transaction.
    MsDb2Command myCommand = myConnection.CreateCommand();
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (1
00, 'Description')";
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (1
01, 'Description')";
        myCommand.ExecuteNonQuery();
```

```
            myTrans.Commit();
            Console.WriteLine("Both records are written to database.");
        }
        catch(Exception e)
        {
            try
            {
               myTrans.Rollback();
            }
            catch (MsDb2Exception ex)
            {
                if (myTrans.Connection != null)
                {
                    Console.WriteLine("An exception of type " + ex.GetType() +
                        " was encountered while attempting to roll back the transaction.");
                }
            }

            Console.WriteLine("An exception of type " + e.GetType() +
                "was encountered while inserting the data.");
            Console.WriteLine("Neither record was written to database.");
        }
        finally
        {
            Console.ReadLine();
            myConnection.Close();
        }
    }
    // End TransactionConnection
```

See Also

Other Resources

Using the Managed Provider for DB2

How to Perform a Two-Phase Commit Transaction over TCP/IP

Two-phase commit (2PC) is a host server-installed protocol that ensures that updates to multiple instances of a database on a network either succeed or fail in their entirety. Host Integration Server 2009 supports 2PC over TCP/IP, enabling you to gain the security of a 2PC connection over the Internet.

Host Integration Server supports 2PC works using three components: the Microsoft Distributed Transaction Coordinator (DTC), the Resync service, and the transaction log. The DTC governs the normal DTC transaction flow: enlist, prepare, commit, and abort. The Resync service coordinates transaction recovery in case of any failure or disconnection, while the transaction log maintains a log of information that is needed in case of recovery.

You can perform a 2PC transaction with ADO.NET and the Managed Provider for DB2 by using the **System.EnterpriseServices** namespace. Using a 2PC transaction is automatic. However, you may need to configure your 2PC connection by using tools such as the Data Access Tool.

Example

The following code example demonstrates how to use 2PC in a DB2 transaction.

```
Public class CentralBank : ServicedComponent
{
   public CentralBank()
   {
   }

   public void Transfer(string connString, int fromId, int toId, double amount)
   {
      MsDb2Connection conn = null;
      MsDb2Command cmd = null;
      int rowsAffected = 0;
      try
      {
         conn = new MsDb2Connection(connString);
         conn.Open();

         cmd = new MsDb2Command();
         cmd.Connection = conn;
         cmd.CommandText = "UPDATE ACCOUNTS SET BALANCE = BALANCE + ?
WHERE ID = ?";
         cmd.Parameters.Add("balance", amount);
         cmd.Parameters.Add("toId", toId);
         rowsAffected = cmd.ExecuteNonQuery();
         cmd.Parameters.Clear();
         cmd.CommandText = "UPDATE ACCOUNTS SET BALANCE = BALANCE - ?
WHERE ID = ?";
         cmd.Parameters.Add("balance", amount);
         cmd.Parameters.Add("fromId", fromId);
         rowsAffected = cmd.ExecuteNonQuery();
         ContextUtil.SetComplete();
      }
      catch(MsDb2Exceptionsqlca)
      {
         Console.WriteLine(sqlca.Message);
         ContextUtil.SetAbort();
      }
      finally
      {
         if(cmd != null)
            cmd.Dispose();
         if(conn != null)
         {
            if(conn.State == ConnectionState.Open)
               conn.Close();
            conn.Dispose();
         }
      }
```

```
        }
    }
```

See Also

Other Resources

Using the Managed Provider for DB2

Obtaining Schema Information from the Managed Provider for DB2

You can obtain schema information from a database by using *schema discovery*. Schema discovery enables applications to request that managed providers find and return information about the database schema, also known as *metadata*, of a given database. Different database schema elements such as tables, columns, and stored-procedures are exposed through schema collections. Each schema collection contains a variety of schema information specific to the provider that is being used.

The Managed Provider for DB2 implements **MsDb2Connection.GetSchema**. The schema information returned from **GetSchema** comes in the form of a **DataTable** object. **GetSchema** provides optional parameters for specifying the schema collection to return, and restricting the amount of information returned.

In This Section

Working with the Managed Provider for DB2 GetSchema Methods

Understanding the Schema Collections for the Managed Provider for DB2

Working with the Managed Provider for DB2 GetSchema Methods

The **Connection** classes in the Managed Provider for DB2 implement a **GetSchema** method, which is used to retrieve schema information about the database that is currently connected, and the schema information returned from the **GetSchema** method comes in the form of a **DataTable** object. The **GetSchema** method is an overloaded method that provides optional parameters for specifying the schema collection to return, and restricting the amount of information that is returned.

Specifying the DB2 Schema Collections

The first optional parameter of the **GetSchema** method is the collection name, which is specified as a string. There are two types of schema collections: common schema collections that are common to all providers, and specific schema collections, which are specific to each provider.

To determine the list of supported schema collections

1. Call **GetSchema** to determine a list of supported schema collections.

   You can call **GetSchema** with no arguments, or with the schema collection name "MetaDataCollections. This returns a **DataTable** object with a list of the supported schema collections, the number of restrictions that they each support, and the number of identifier parts that they use.

Specifying the Restriction Values for a DB2 Schema Collection

The second optional parameter of the **GetSchema** method is the restrictions that are used to limit the amount of schema information returned, and it is passed to the **GetSchema** method as an array of strings. The position in the array determines the values that you can pass, and this is equivalent to the restriction number.

Note
The number of elements in the array must be less than or equal to the number of restrictions supported for the specified sch ema collection, or an **ArgumentException** is thrown. There can be fewer than the maximum number of restrictions. The mis sing restrictions are assumed to be null (unrestricted).

To determine the list of supported restrictions

1. Call the **GetSchema** method with the name of the restrictions schema collection, which is "Restrictions".

   This returns a **DataTable** object with a list of the collection names, restriction names, default restriction values, and restriction numbers.

See Also

Other Resources

Obtaining Schema Information from the Managed Provider for DB2

Understanding the Schema Collections for the Managed Provider for DB2

The following tables describe the schema collections that are implemented by the Managed Providers for DB2. You can query the Managed Provider for DB2 to determine the list of supported schema collections by calling the **GetSchema** method with no arguments, or with the schema collection name "MetaDataCollections". This returns a **DataTable** object with a list of the supported schema collections, the number of restrictions that they each support, and the number of identifier parts that they use. These collections describe all of the required columns. If a provider cannot determine the value of a required column, it returns **null**.

Common Schema Collections

- MetaDataCollections
- DataSourceInformation
- DataTypes
- Restrictions
- ReservedWords

Provider-Specific Collections

- Tables
- Columns
- Procedures
- ProcedureParameters
- Incexes
- PrimaryKeys
- ForeignKeys
- TablePrivileges
- Statistics

Example
See Also
Other Resources
Obtaining Schema Information from the Managed Provider for DB2

Managed Provider for DB2 Tutorial

The following tutorial shows how to configure a connection to a DB2 database and build a simple XML Webservice to expose DB2 tables using ASP.NET. The sample also shows how to retrieve a recordset of information from a DB2 table and return that recordset as a .NET DataSet via an XML Webservice.

In This Section

Getting Started with the Managed Provider for DB2 Tutorial

Step 1: Configuring a Connection to DB2

Step 2: Creating an XML Web Service

Step 3: Adding a Web Service to the Project

Step 4: Executing the Web Service

Reference

Microsoft.HostIntegration.DataAccessLibrary

Microsoft.HostIntegration.MsDb2Client

See Also

Other Resources

Managed Provider for DB2 Programmer's Guide

Getting Started with the Managed Provider for DB2 Tutorial

Before you start the Managed Provider for DB2 tutorial, make sure to perform the following actions:

1. Install Visual Studio 2005
2. Ensure IIS is installed and ASP.NET 2.0 is configured.
3. Install Host Integration Server.
4. Ensure you have access to a DB2 instance with permissions to query and modify data.

See Also

Tasks

Step 1: Configuring a Connection to DB2

Other Resources

Managed Provider for DB2 Tutorial

Step 1: Configuring a Connection to DB2

The first step you need to perform is to create and configure a connection string to your DB2 data source. Once you are finished creating the string, you will use the string to create an XML web service.

To create the connection string

1. Start the Data Access Tool by clicking on **Start**, then **Programs**, then **Microsoft Host Integration Server 2009**, and then click **Data Access Tool**.
2. On the Data Access Tool, click **File**, select **New**, click **Data Source...**, and then click **Next**.
3. On the Data Source page, select your data source platform and network type, and then click **Next**.

   For the purpose of this tutorial, we will select DB2/MVS from **Data source platform**, and TCP/IP as a **Network type**.
4. On the TCP/IP Network Connection page, enter the IP address or alias of your DB2 source into the **Address or alias** field, and enter the port number of your DB2 source into the **Port** field.
5. Ensure the **Distributed Transactions** box is unchecked, and then click **Next**.
6. On the DB2 Database page, fill in the **Initial Catalog**, **Package collection**, **Default schema**, and **Default qualifier** fields with the information relevant to your database, and then click **Next**.
7. On the Locale page, confirm that the locales are correct for your system, and then click **Next**.
8. On the Security page, select Interactive Sign-On from the **Security Method** drop-down box, enter your user name and password in the **Properties** fields, and then click **Next**.
9. On the Advanced Options page, confirm that no options are checked, and then click **Next**.
10. On the Validation page, click **Connect** to connect to your database using the information you provided, and click **Sample Query** to send a sample query to your DB2 database.

    Once you confirm that your information is correct, click **Next**.
11. On the Saving Information page, check the **Initialization string file** box, enter a data source name in the **Data source name** field, and then click **Next**.
12. On the Completing the Data Source Wizard, click **Finish**.

To view the connection string

1. Click **Start**, and then click **Run...**.
2. In the **Run** dialog, type **Notepad** in the **Open:** field, and then click **OK**.
3. In Notepad, click **File**, and then click **Open...**.
4. Use the Open dialog to browse to the .txt file that contains the connection string information for the data source.

   For this tutorial, the file name is HIS_TRAINING.txt, and is located in c:\Documents and Settings\username\My Documents\Host Integration Projects\Data Sources.

See Also

Tasks

Step 2: Creating an XML Web Service

Other Resources

Managed Provider for DB2 Tutorial

Step 2: Creating an XML Web Service

Once you have created the connection string to your DB2 database, you can create the XML web service. Once you have created the service, you can add the service to the project.

To create the XML web service

1. In Visual Studio, click File, select New, and then click Project...
2. In the New Project dialog, in the **Project types:** pane, click on Other Project Types, and then select Visual Studio Solutions.
3. In the **Templates** pane, confirm that Blank Solution is selected.
4. Type **HISTraining** in the **Name** field, and then click **OK**.

To add code to the web service

1. In Solution Explorer, right-click HISTraining, select **Add...**, and then click **New Project**.
2. In the Add New Project dialog, in the **Project types:** pane, click **Other Languages**, and then select **Visual Basic**.
3. In the **Templates:** pane, select **Class Library**.
4. In the **Name** field, type **DB2DAL**, and then click **OK**.
5. In Solution Explorer, right-click DB2DAL, select **Add...**, and then click **Add Reference...**.
6. In the Add Reference dialog, on the **.NET** tab, select Microsoft.HostIntegration.MsDb2Client, and then click **OK**.
7. Add the following code to your Class1.vb file:

```
Imports Microsoft.HostIntegration.MsDb2Client
Public Class DB2DAL
    Public Function executeSQL(ByVal sqlQuery As String, ByVal connString As String) A
s DataSet
        Dim db2Conn As New MsDb2Connection(connString)
        Dim db2Cmd As New MsDb2Command(sqlQuery, db2Conn)
        Dim db2DA As New MsDb2DataAdapter(db2Cmd)
        Dim returnDS As New DataSet
        db2Conn.Close()
        db2DA.Fill(returnDs)
        db2Conn.Close()
        Return returnDS
    End Function
End Class
```

8. On the **Build** menu, click **Build Solution**.
9. On the **File** menu, click **Save All**.

See Also

Tasks

Step 3: Adding a Web Service to the Project

Other Resources

Managed Provider for DB2 Tutorial

Step 3: Adding a Web Service to the Project

Once you have created the XML web service, you can add the web service to your project. Once you have added the web service, you can start the application.

To add a web service to the project

1. In the Solution Explorer, right-click HISTraining, select **Add...**, and then click **New Web Site**.
2. In the Add New Web Site dialog, select ASP.NET Web Service.
3. In the Location drop-down box, select HTTP, and enter **http://localhost/DB2WebService** to the field, and then click OK
4. In Solution Explorer, right-click **http://localhost/DB2WebService**, and then click **Add Reference...**.
5. In the Add Reference dialog, on the **Projects** tab, select DB2DAL, and then click OK.
6. In Solution Explorer, expand **http://localhost/DB2WebService**, expand App_Code, and double-click Service.vb.
7. Add the following code to Service.vb

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function executeSQLQuery(ByVal connstring As String, ByVal sqlStatement As
String) As Data.DataSet
        Dim dal As New DB2DAL.DB2DAL
        Return dal.executeSQL(sqlStatement, connstring)
    End Function

End Class
```

8. On the **Build** menu, click **Build Solution**.
9. On the File menu, click **Save All**.

See Also

Tasks

Step 4: Executing the Web Service

Other Resources

Managed Provider for DB2 Tutorial

Step 4: Executing the Web Service

Once you have added the web service to your project, you can execute the application.

To run the Managed DB2 tutorial

1. In Solution Explorer, double-click on Service.asmx
2. Right-click in the Service.asmx pane, and select **View in Browser**.
3. On the Service page, click **executeSQLQuery**.
4. On the ExecuteSQLQuery page, enter the connection string created in Step 1 into the **connstring:** field.
5. In the **squStatement:** field, type **SELECT * FROM AREAS**, and then click **Invoke**.

See Also

Other Resources

Managed Provider for DB2 Tutorial

Managed Data Provider for Host Files Programmer's Guide

This section describes the general architecture and implementation details of the Managed Provider for Host Files. The Managed Provider for Host Files enables a client to connect to a remote host file system, read and write data from the host file system using a managed framework, exposes bulk file transfer, and enables remote commands to the host.

In This Section

Managed Data Provider for Host Files

This section contains a discussion about the background and general design principles surrounding the Managed Data Provider for Host Files.

In This Section

Goals for the Managed Data Provider for Host Files

The primary purpose of the Managed Data Provider for Host Files is to enable developers to access files and data structures on mid-range and mainframe systems using the Microsoft .NET Framework. This differs from the Managed Provider for DB2, which enables developers to access host DB2 recordsets.

The Managed Data Provider for Host files was designed with the following capabilities in mind:

- Granting developers access to the file system commands of mid-range and mainframe systems. By exposing the file system commands, a developer can perform such activities as setting record attributes, locking files and records, navigating between records, and modifying file contents. Such activities are exposed primarily through the Managed Provider for Host Files interface.

- Granting access to the file commands of the most popular mid-range and mainframe file server systems. In doing so, developers familiar with AS/400 input and output commands can use their knowledge to access mid-range file systems. This differs from the Managed Provider for DB2, which allows developers familiar with SQL to use that knowledge to access DB2/400 tables.

- Improving the bulk transfer rate between systems.

See Also
Other Resources
Managed Data Provider for Host Files

Using the Managed Data Provider For Host Files

The Managed Data Provider for Host Files operates in most ways as a normal data provider: you can connect to a host file system, execute commands, retrieve data, and use stored procedures.

In This Section

How to Create an Application Using the Managed Data Provider for Host Files

The following topics describe the basic setup for creating and deploying an application that uses the Managed Data Provider for Host Files.

In This Section

How to Create a Project in Visual Studio for the Managed Provider for Host Files

How to Create an Assembly for the Managed Data Provider for Host Files

How to Deploy an Assembly for the Managed Data Provider for Host Files

See Also

Other Resources

Using the Managed Data Provider For Host Files

How to Create a Project in Visual Studio for the Managed Provider for Host Files

Host Integration Server 2009 provides a specific way to create a project that uses the Managed Provider for Host files.

To create a project in Visual Studio using the Managed Provider for Host Files

1. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, click **Host Integration Projects**.
3. In the **Templates** pane, click **Host File Project**.
4. In the **Name** field, type the name of the file.
5. In the **Location** field, type the location to save the project.
6. Click **OK** to create the project.

See Also

Other Resources

Using the Managed Data Provider For Host Files

BizTalk Adapter for Host Files

How to Create an Assembly for the Managed Data Provider for Host Files

Once you have finished defining a remote environment for your application, you can use Visual Studio to create the application assembly. Host Integration Server provides a variety of wizards and tools to help you in this process:

1. Create a Host File project.
2. Add one or more .NET client libraries to the project that will contain your application.
3. Create the interfaces for your application using Host Integration Server Designer.

   HIS Designer enables you to create interfaces that represent data and methods native to your host environment. By programming towards these interfaces, you can pass information and commands through Host Integration Server and onto the mainframe.

After you create the assembly, you can deploy your assembly to your local computer and test the interfaces against your code.

To create an application assembly that uses the Managed Provider for Host Files

1. Start Visual Studio.
2. Click **File**, then **New**, and then click **Project**.
3. In the **New Project** dialog box, in the **Project types:** pane, click on Host Integration Projects.
4. In the **Templates** pane, click **Host File Project**.
5. In the **Name** field, type the name of your project.
6. In the **Location** field, type the location to save your project, and then click **OK**.

To add an assembly to your project

1. Click **Project**, and then click **Add .NET Client Library**.

   The Managed Provider for Host Files supports .NET only.
2. On the **Add New Item** dialog box, in the **Templates** pane, confirm that .NET Client Library is highlighted.
3. In the **Name:** field, type the name of the assembly, and then click **Add**.
4. On the **Welcome to the .NET Client Library Wizard** page, click **Next**.
5. On the **Completing the .NET Client Library Wizard** page, confirm that the displayed settings are correct, and then click **Create**.

To define interfaces using the HIS Designer

1. If you do not have a Host Definition file (.hcd) file available, you can use the **Import COBOL Wizard** or the **Import RPG Wizard** to define your interfaces.

   For more information, see How to Import COBOL into a TI Component.
2. If you have a previous .NET client library that you want to base your new object on, you can use the Import Library tool to import the library into your project.

   For more information, see How to Import a TI Component.
3. If you want to manually create or modify the interface definitions, you can do so using the console tree of HIS Designer.

See Also
Concepts
Creating an Application for the BizTalk Adapter for Host Applications

How to Deploy an Assembly for the Managed Data Provider for Host Files

Once you have created the assembly that contains your code, you can deploy the assembly to your local computer. Deploying the assembly registers the assembly with Host Integration Server and Transaction Integrator, and loads the assembly into the appropriate location to be used. After you deploy the assembly, you can test the interfaces and the code that you wrote. You can then undeploy the assembly, modify the code, and redeploy as necessary.

To deploy and undeploy an assembly to your local machine

1. In HIS Designer, select the tab that has the name of the assembly to deploy.
2. In the **Properties** window, confirm that you have selected the remote environment that you want your assembly to communicate with.
3. In the HIS tree node, right-click the name of the assembly, and select **Deploy**.
4. Test your application as necessary.
5. When you are finished, you may undeploy your assembly by right-clicking the assembly name in HIS Designer, and selecting **UnDeploy**.

See Also

Concepts

Creating an Application for the BizTalk Adapter for Host Applications

How to Connect to and Disconnect from a Host File System

The first step in accessing a host file system is to connect to the file system. You must use **HostFileConnection** to access the host file system. After you have connected, you can retrieve, modify, and update the information that you want.

Procedure
To connect and disconnect to a Host File System

1. Create a **HostFileConnection** object, using the connection string that describes the Host system.
2. Open a connection with a call to **HostFileConnection.Open**.
3. Interact with the host file system as necessary.
4. When you are finished, release the **HostFileConnection** object with a call to **HostfileConnection.Close** and **HostFileConnection.Dispose**.

   Connections are not implicitly released when a **HostFileConnection** falls out of scope. Therefore, you need to release the object when you are finished using it.

Example

The following very simple code example shows how to use **HostFileConnection** to open and close a connection.

```
try
   {
      HostFileConnection cn = new HostFileConnection();
      cn.ConnectionString = cnstring;
      cn.Open();
      // Perform tasks here.
      cn.Close();
      cn.Dispose();
   }
```

See Also
Other Resources
Using the Managed Data Provider For Host Files
BizTalk Adapter for Host Files

How to Execute Commands in the Host File System

After establishing a connection to a data source, you can execute commands and return results from the data source using **HostFileCommand**.

Important
The Managed Provider for Host Files does not support any type of transaction. Therefore, you should try to avoid using INSERT, UPDATE, or DELETE commands on mission-critical data.

Procedure
To execute a command on the host file system

1. Establish a connection using **HostFileConnection**.

   For more information, see How to Connect to and Disconnect from a Host File System.

2. Once connected, create a **HostFileCommand** object by using **HostfileConnection.CreateCommand**.

3. Use the **HostFileCommand** object to execute commands on the Host File system.

   HostFileCommand exposes several Execute methods that you can use:

   - When returning results as a stream of data, use **ExecuteDbDataReader** to return a **DataReader** object.
   - Use **ExecuteScalar** to return a singleton value.
   - Use **ExecuteNonQuery** to execute commands that do not return rows.
   - Use **ExecuteRecordSet** to execute commands on a recordset.

Note
When modifying an Alternate Index File (AIX), you may receive an "Invalid record length" error when the Index is defined not to accept duplicate keys. This error may occur because the INDEX of the Alternate Index VSAM file is not large enough to hold multiple key values for the same index record .

See Also
Other Resources
Using the Managed Data Provider For Host Files
BizTalk Adapter for Host Files

Retrieving Information from the Host File System

The topics in this section describe how to retrieve information from the Host File System.

In This Section

How to Retrieve Data from the Host File System

How to Retrieve Multiple Resultsets from the Host File System

How to Retrieve Schema Sets from the Host File System

See Also

Other Resources

Using the Managed Data Provider For Host Files

BizTalk Adapter for Host Files

How to Retrieve Data from the Host File System

Just as you can do with other managed data providers, you can access host data with an implementation of a **DataReader** object through **HostfileCommand**.

To retrieve data using a data reader

1. Create an instance of **HostFileCommand**.
2. Create a **DataReader** object through a call to **HostFileCommand.ExecuteDBDataReader**.

   Calling **ExecuteDBDataReader** retrieves data rows from the data source.
3. Use **DBDataReader.Read** to obtain a row from the results of the query.

   You can access each column of the returned row by passing the name or ordinal reference of the column to the **DBDataReader** object. However, for best performance, the **DBDataReader** object provides a series of methods that enable you to access column values in their native data types (**GetDateTime**, **GetDouble**, **GetGuid**, **GetInt32**, and so on).
4. Once you are finished with the **DBDataReader** object, call **DBDataReader.Close**.

   If your **HostFileCommand** object contains output parameters or return values, they will not be available until the **DBDataReader** is closed.

   Note that while **DBDataReader** is open, the **HostFileConnection** is in use exclusively by that **DBDataReader**. You cannot execute any commands for the **HostFileConnection**, including creating another **DBDataReader**, until the original **DBDataReader** is closed.

See Also

Other Resources

Retrieving Information from the Host File System

BizTalk Adapter for Host Files

How to Retrieve Multiple Resultsets from the Host File System

If multiple result sets are returned, the DBDataReader class provides the **NextResult** method to iterate through the result sets in order.

Example
See Also
Other Resources
Retrieving Information from the Host File System
BizTalk Adapter for Host Files

How to Retrieve Schema Sets from the Host File System

When a **HostFileConnection** is open, you can retrieve schema information about the target data by using the **GetSchema** method. **GetSchema** returns a **DataTable** object populated with the rows and columns that contain the schema information of the target of the current connection.

In addition, while a **DBDataReader** is open, you can retrieve schema information about the current result set by using the **GetSchemaTable** method. **GetSchemaTable** returns a **DataTable** object populated with rows and columns that contain the schema information for the current result set. The **DataTable** object contains one row for each column of the result set. Each column of the schema table row maps to a property of the column returned in the result set, where the **ColumnName** is the name of the property and the value of the column is the value of the property.

To retrieve schema sets from the host file system

1. Open a connection to the host file system with a call to **HostFileConnection**.
2. Call **HostfileConnection.GetSchema** to retrieve the schema data.

Example

The following code example demonstrates how to retrieve the schema sets from a connection object. Note that the ETCMLogging and HostFileUtils objects are developer-created objects that provide logging and utility functionality.

```
public void CNGetSchema(ETCMLogging.Logging logging, string host, string ccsid, string cnst
ring, HostFileUtils.Utils.HostFileType hostfiletype)
        {
            HostFileUtils.Oledb oledb = new HostFileUtils.Oledb();
            HostFileUtils.Utils u = new HostFileUtils.Utils();
            logging.LogInfo(host);
            try
            {
                // Create connection.
                HostFileConnection cn = oledb.CreateConnection(logging);
                cn.ConnectionString = cnstring;
                DataTable dt = cn.GetSchema();
                if (dt.HasErrors)
                {
                    logging.LogFail("returned datatable has errors");
                }
                // Open the connection.
                logging.LogInfo("Open Connection");
                cn.Open();
                DataTable dt2 = cn.GetSchema();
                if (dt2.HasErrors)
                {
                    logging.LogFail("returned datatable has errors");
                }
                int rowcnt = dt.Rows.Count;
                for (int i = 0; i < rowcnt; i++)
                {
                    int colcnt = dt.Rows[i].ItemArray.Length;
                    for (int o = 0; o < colcnt; o++)
                    {
                        u.CompareValues(dt.Rows[i][o].ToString(), dt2.Rows[i][o].ToString()
, logging);
                    }
                }
                // Close the open connection.
                cn.Close();
            }
            catch (Exception e)
            {
                logging.LogInfo(e.Message);
                logging.LogFail(e.StackTrace);
            }
        }
```

See Also

Other Resources

Retrieving Information from the Host File System

BizTalk Adapter for Host Files

How to Obtain a Single Value from a Host File System

You might need to return database information that is just a single value rather than in the form of a table or data stream. For example, you might want to return the result of an aggregate function such as COUNT(*), SUM(Price), or AVG(Quantity).

To obtain a single value from the Host File System

1. Make a call to **HostFileCommand.ExecuteScalar**.

   ExecuteScalar returns a scalar value: the value of the first column of the first row of the result set.

See Also

Other Resources

Using the Managed Data Provider For Host Files

BizTalk Adapter for Host Files

Working with the Host File Adapter and Dataset

A **HostFileDataAdapter** is used to retrieve data from a data source and populate tables within a dataset. The **HostFileDataAdapter** also resolves changes made to the dataset back to the data source. The **HostFileDataAdapter** uses the **HostFileConnection** object to connect to a data source, and it uses **HostFileCommand** objects to retrieve data from and resolve changes to the data source.

In This Section

How to Populate a Host File Dataset from the Data Adapter

How to Update the Host File System with the Data Adapter

How to Add Constraints to the Host File Dataset

How to Close a Connection with the Host File Adapter

See Also

Other Resources

Using the Managed Data Provider For Host Files

BizTalk Adapter for Host Files

How to Populate a Host File Dataset from the Data Adapter

The dataset is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The dataset represents a complete set of data including tables, constraints, and relationships among the tables. Because the dataset is independent of the data source, a dataset can include data local to the application, and also data from multiple data sources. Interaction with existing data sources is controlled through the **DataAdapter** object.

The **HostfileDataAdapter.SelectCommand** property is a **HostFileCommand** object that retrieves data from the data source. The **HostFileDataAdapter.Fill** method is used to populate a dataset with the results of the **SelectCommand**. **Fill** takes as its arguments a **DataSet** object to be populated, and a **DataTable** object, or the name of the **DataTable** to be filled with the rows returned from the **SelectCommand**.

The **Fill** method uses the **HostFileDataReader** object implicitly to return the column names and types used to create the tables in the **DataSet** object, and also the data to populate the rows of the tables in the **DataSet** object. Tables and columns are only created if they do not already exist; otherwise **Fill** uses the existing **DataSet** schema. Primary keys are not created unless they are in the data source, and **HostFileDataAdapter.MissingSchemaAction** is set to **MissingSchemaAction.AddWithKey**. If **Fill** finds that a primary key exists for a table, it overwrites data in the **DataSet** object with data from the data source for rows where the primary key column values match those of the row returned from the data source. If no primary key is found, the data is appended to the tables in the **DataSet** object. **Fill** uses any mappings that might exist when populating the **DataSet** object.

If the **HostFileDataAdapter** encounters multiple result sets, it creates multiple tables in the **DataSet** object. The tables are given an incremental default name of TableN, starting with "Table" for Table0. If a table name is passed as an argument to the **Fill** method, the tables are given an incremental default name of TableNameN, starting with "TableName" for TableName0.

You can use any number of **HostFileDataAdapter** objects with a **DataSet** object. Each **DataAdapter** object can be used to fill one or more **DataTable** objects and resolve updates back to the relevant data source. You can add **DataRelation** and **Constraint** objects to the **DataSet** locally, enabling you to relate data from dissimilar data sources. One or more **DataAdapter** objects can handle communication to each data source.

To populate a host file dataset from the data adapter

1. Create a new connection to your data source by using **HostFileConnection**.
2. Open the connection by using **HostFileConnection.Open**.
3. Create a SELECT command that describes the data to retrieve with **HostFileCommand**.
4. Create a **HostFileDataAdapter** using **HostFileConnection** to interact with the stored data.
5. Create a **DataSet** object to store the data locally.
6. Retrieve the data through the **HostFileDataAdapter** using the **DataSet** object and the **Fill** command.

Example

The following code example shows how to fill a dataset through a **HostFileDataAdapter**. In this example, the ETCMLogging and HostFileUtils objects supply logging and utility functionality, respectively.

```
public void HFDAdapterCommandConstructor(ETCMLogging.Logging logging, string host, string c
csid, string cnstring, HostFileUtils.Utils.HostFileType hostfiletype)
        {
            HostFileUtils.Utils u = new HostFileUtils.Utils();
            logging.LogInfo(host + "::" + hostfiletype.ToString());
            HostFileUtils.Utils.MytestsVals[] Datavals = u.InitMytestsVals();

            try
            {
                HostFileConnection cn = new HostFileConnection(cnstring);
                cn.Open();
                String SELECT = u.CreateSQLCommand(host, hostfiletype, cnstring, "SELECT",
"MYTEST");
                HostFileCommand hfc = new HostFileCommand(SELECT, cn);
                HostFileDataAdapter hfda = new HostFileDataAdapter(hfc);
                DataSet ds = new DataSet();
                hfda.Fill(ds);
                int[] cp = u.CheckColumns(SELECT, cn, logging);
```

```
                u.ValidateDataSet(ds, logging, Datavals, cp);

                cn.Close();
            }
            catch (Exception e)
            {
                logging.LogInfo(e.Message);
                logging.LogFail(e.StackTrace);
            }
        }
```

In this code example, the **HostFileUtils** object and the *cnstring* and *ccsid* parameters enable you to quickly create a test SQL command with the relevant information.

See Also

Other Resources

Working with the Host File Adapter and Dataset

BizTalk Adapter for Host Files

How to Update the Host File System with the Data Adapter

HostFileDataAdapter.Update is called to resolve changes from a **DataSet** object back to the data source. The **Update** method, like the **Fill** method, takes an instance of a **DataSet** as an argument.

To update the host file system with the data adapter

1. Create a **DataSet** object that contains the information that you want to update.

   Or, you can overwrite the data of an existing **DataSet** object with a call to **DataSet.AcceptChanges**.

   Note that calling **AcceptChanges** on the **DataSet**, **DataTable**, or **DataRow** object causes all Original values for a **DataRow** object to be overwritten with the Current values for the **DataRow**. If the field values that identify the row as unique have been modified, after you call **AcceptChanges**, the Original values no longer match the values in the data source.

   In addition, you can use **HostFileCommand** parameters to specify input and output values for a SQL statement for each modified row in a **DataSet** object.

2. Call **HostFileDataAdapter.Update**, with the DataSet object that you want to update.

   When you call the **Update** method, the **HostFileDataAdapter** analyzes the changes that have been made and executes the appropriate command. If **Update** is called and the appropriate command does not exist for a particular update (for example, no **DeleteCommand** for deleted rows), an exception is thrown.

3. If you want to update your dataset with data, call **HostFileDataAdapter.Fill** on your **DataSet** object.

   The **Update** method resolves your changes back to the data source; however other clients may have modified data at the data source since the last time you filled the DataSet. New rows are added to the table, and updated information is incorporated into existing rows.

Example

The following code example demonstrates how to updating a dataset with the **Fill** and **Update** commands. Note that the ETCMLogging and HostFileUtils objects provide logging and utility functionality, respectively.

```
public void BVTHFDataAdapterInsert(ETCMLogging.Logging logging, string host, string ccsid,
string cnstring, HostFileUtils.Utils.HostFileType hostfiletype)
        {
            HostFileUtils.Utils u = new HostFileUtils.Utils();
            logging.LogInfo(host + "::" + hostfiletype.ToString());
            HostFileUtils.Utils.BvttestsVals[] Datavals = u.InitBvttestsVals();

            try
            {
                HostFileConnection cn = new HostFileConnection(cnstring);
                cn.Open();
                String SELECT = u.CreateSQLCommand(host, hostfiletype, cnstring, "SELECT",
"BVTTESTS");
                HostFileDataAdapter hfda = new HostFileDataAdapter(SELECT, cn);
                DataSet ds = new DataSet();
                DataSet dsold = new DataSet();
                hfda.Fill(ds);
                hfda.Fill(dsold);
                int[] cp = u.CheckColumns(SELECT, cn, logging);
                u.ValidateDataSet(ds, logging, Datavals, cp);
                object[] newrow = new object[5];
                // ('REC129-1','REC129-2',129,1290,'129.645')
                newrow[cp[0]] = "REC129-1";
                newrow[cp[1]] = "REC129-2";
                newrow[cp[2]] = 129;
                newrow[cp[3]] = 1290;
                newrow[cp[4]] = 129.645M;
                ds.Tables[0].Rows.Add(newrow);
                int z = hfda.Update(ds);
                if (z != 1)
                {
```

```
                    logging.LogFail("a unexpected number of updates::"+z.ToString());
                }
                DataSet ds1 = new DataSet();
                hfda.Fill(ds1);
                int j = 0;
                int i = 0;
                foreach (DataRow row in ds1.Tables[0].Rows)
                {
                    string rec = (string)ds1.Tables[0].Rows[j][cp[0]];
                    if (!rec.Equals("REC129-1"))
                    {
                        u.CompareValues((string)ds1.Tables[0].Rows[j][cp[0]], Datavals[i].O
UT1_CHAR1, logging);
                        u.CompareValues((string)ds1.Tables[0].Rows[j][cp[1]], Datavals[i].O
UT1_CHAR2, logging);
                        u.CompareValues((short)ds1.Tables[0].Rows[j][cp[2]], Datavals[i].OU
T1_SMALLINT, logging);
                        u.CompareValues((int)ds1.Tables[0].Rows[j][cp[3]], Datavals[i].OUT1
_INTEGER, logging);
                        u.CompareValues((decimal)ds1.Tables[0].Rows[j][cp[4]], Datavals[i].
OUT1_DECIMAL, logging);
                        j++;
                        i++;
                    }
                    else
                    {
                        u.CompareValues((string)ds1.Tables[0].Rows[j][cp[0]], "REC129-1", l
ogging);
                        u.CompareValues((string)ds1.Tables[0].Rows[j][cp[1]], "REC129-2", l
ogging);
                        u.CompareValues((short)ds1.Tables[0].Rows[j][cp[2]], 129, logging);
                        u.CompareValues((int)ds1.Tables[0].Rows[j][cp[3]], 1290, logging);
                        u.CompareValues((decimal)ds1.Tables[0].Rows[j][cp[4]], 129.645M, lo
gging);
                        j++;
                    }
                }
                if (j == 0)
                {
                    logging.LogFail("No Rows on DataTable!");
                }
                z = 0;
                z = hfda.Update(dsold);
                if (z != 1)
                {
                    logging.LogFail("a unexpected number of updates::" + z.ToString());
                }
                DataSet ds2 = new DataSet();
                hfda.Fill(ds2);
                u.ValidateDataSet(ds2, logging, Datavals, cp);

                cn.Close();
            }
            catch (Exception e)
            {
                logging.LogInfo(e.Message);
                logging.LogFail(e.StackTrace);
            }
        }
```

See Also

Other Resources

Working with the Host File Adapter and Dataset

BizTalk Adapter for Host Files

How to Add Constraints to the Host File Dataset

The **HostFileDataAdapter.Fill method** fills a **DataSet** object with table columns and rows from a data source; though constraints are commonly set by the data source, the **Fill** method does not add this schema information to the **DataSet** object by default. To populate a **DataSet** object with existing primary key constraint information from a data source, you can call **HostFileDataAdapter.FillSchema**.

Note
If a **column** in a data source is identified as auto-incrementing, the **FillSchema** method, or the **Fill** method with a **MissingSchemaAction** of **AddWithKey**, creates a **DataColumn** that has an **AutoIncrement** property set to **true**. However, you must set the **AutoIncrementStep** and **AutoIncrementSeed** values yourself.

To populate a dataset with additional key constraints

1. Call **HostFileDataAdapter.FillSchema**, using the targeted **DataSet** and Schema that contains the specified key constraints.

   Adding schema information to a **DataSet** before filling it with data ensures that primary key constraints are included with the **DataTable** objects in the **DataSet** object. As a result, when additional calls to fill the **DataSet** are made, the primary key column information is used to match new rows from the data source with current rows in each **DataTable** object, and current data in the tables is overwritten with data from the data source. Without the schema information, the new rows from the data source are appended to the **DataSet** object, resulting in duplicate rows.

See Also

Other Resources

Working with the Host File Adapter and Dataset

BizTalk Adapter for Host Files

How to Close a Connection with the Host File Adapter

If you create a HostFileDataAdapter object with a connection string, the object will automatically create a connection object. Once you are finished using a host file adapter, you need to dispose of the implicit connection you made. You can use the Dispose and Close commands to do so.

To close the connection created implicitly through a HostFileDataAdapter object

1. Once you are finished with the connection, call HostFileDataAdapter.Dispose() to dispose of the connection.
2. Alternately, you may also call HostFileDataAdapter.SelectCommand.Connection.Close() to close the connection as well.

Example

The following code sample shows how to create a connection through a HostFileDataAdapter object, and how to properly dispose of the connection.

```
try
{
    HostFileDataAdapter hfda = new HostFileDataAdapter(SELECT,"valid connection string");
    DataSet ds = new DataSet();
    hfda.Fill(ds);
    string xml = ds.GetXml();
    Console.WriteLine(xml);
    hfda.Dispose();
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

See Also

Other Resources

Working with the Host File Adapter and Dataset

Obtaining Schema Information from the Host File System

You can obtain schema information from a database by using *schema discovery*. Schema discovery enables applications to request that managed providers find and return information about the database schema, also known as *metadata*, of a given database. Different database schema elements such as tables, columns, and stored procedures are exposed through schema collections. Each schema collection contains a variety of schema information specific to the provider that is being used.

The Managed Provider for Host Files implements **HostFileConnection.GetSchema** class, and the schema information that is returned from **GetSchema** comes in the form of a **DataTable** object. **GetSchema** also provides optional parameters for specifying the schema collection to return, and restricting the amount of information returned.

In This Section

Working with the Host File GetSchema Methods

Common Schema Collections for the Host File System

See Also

Other Resources

Using the Managed Data Provider For Host Files

BizTalk Adapter for Host Files

Working with the Host File GetSchema Methods

The Managed Provider for Host File **HostFileConnection** class implements a **GetSchema** method, which is used to retrieve schema information about the file system that is currently connected. The schema information that is returned from the **GetSchema** method comes in the form of a **DataTable** object. The **GetSchema** method is an overloaded method that provides optional parameters for specifying the schema collection to return, and restricting the amount of information that is returned.

To retrieve file system schema information

1. Create a **HostFileConnection** object that represents the connection to the host file system.
2. Retrieve the schema information by calling **HostFileConnection.GetSchema**.
    a. The first optional parameter of the **GetSchema** method is the collection name, which is specified as a string. There are two types of schema collections: common schema collections that are common to all providers, and specific schema collections, which are specific to each provider. You can call **GetSchema** either with no parameters, or else with the schema collection name "MetaDataCollections". This returns a DataTable object with a list of the supported schema collections, the number of restrictions that they each support, and the number of identifier parts that they use.

    b. The second optional parameter of the **GetSchema** method is the restrictions that are used to limit the amount of schema information returned, and it is passed to the **GetSchema** method as an array of strings. The position in the array determines the values that you can pass, and this is equivalent to the restriction number.

3. If you want to put a restriction on the Tables schema collection, consider the following:
    a. Create an array of strings with four elements.

    b. Put a value in the element that matches the restriction number.

        For example, to restrict the tables returned by the **GetSchema** method to only those tables that are owned by the "dbo" role, set the second element of the array to "dbo".

    c. Pass the value into your **GetSchema** call.

To determine a list of supported restrictions on a Schema

1. Call **GetSchema** with the first parameter set to "Restrictions".

    This returns a DataTable object with a list of the collection names, the restriction names, the default restriction values, and the restriction numbers.

See Also
Other Resources
Obtaining Schema Information from the Host File System
BizTalk Adapter for Host Files

Common Schema Collections for the Host File System

The common schema collection is the schema collection that is implemented by the Managed Provider for Host Files. You can query the managed provider to determine the list of supported schema collections by calling the **GetSchema** method with no arguments, or with the schema collection name "MetaDataCollections". This returns a **DataTable** object with a list of the supported schema collections, the number of restrictions that they each support, and the number of identifier parts that they use.

The following tables describe the common schema collections for the Host File System.

Columns

Column Name	Data Type	Description
table_catalog	String	Catalog of the table.
table_schema	String	Schema that contains the table.
table_name	String	Table name.
column_name	String	Column name.
ordinal_position	Int16	Column identification number.
column_default	String	Default value of the column
is_nullable	String	Nullability of the column. If this column allows NULL, this column returns YES. Otherwise, No is returned.
data_type	String	System-supplied data type.
character_maximum_length	Int32 – Sql8, Int16 – Sql7	Maximum length, in characters, for binary data, character data, or text and image data. Otherwise, NULL is returned.
character_octet_length	Int32 – SQL8, Int16 – Sql7	Maximum length, in bytes, for binary data, character data, or text and image data. Otherwise, NULL is returned.
numeric_precision	Unsigned Byte	Precision of approximate numeric data, exact numeric data, integer data, or monetary data. Otherwise, NULL is returned.
numeric_precision_radix	Int16	Precision radix of approximate numeric data, exact numeric data, integer data, or monetary data. Otherwise, NULL is returned.
numeric_scale	Int32	Scale of approximate numeric data, exact numeric data, integer data, or monetary data. Otherwise, NULL is returned.
datetime_precision	Int16	Subtype code for datetime and SQL-92 interval data types. For other data types, NULL is returned.
character_set_catalog	String	Returns master, indicating the database in which the character set is located, if the column is character data or text data type. Otherwise, NULL is returned.
character_set_schema	String	Always returns NULL.
character_set_name	String	Returns the unique name for the character set if this column is character data or text data type. Otherwise, NULL is returned.

collation_catalog	String	Returns master, indicating the database in which the collation is defined, if the column is character data or text data type. Otherwise, this column is NULL.

DataSourceInformation

CompositeIdentifierSeparatorPattern	string	The regular expression to match the composite separators in a composite identifier. For example, "\." (for SQL Server) or "@|\." (for Oracle).
		A composite identifier is typically what is used for a database object name, for example: pubs.dbo.authors or pubs@dbo.authors.
		For SQL Server, use the regular expression "\.". For OracleClient, use "@|\.".
		For ODBC use the Catalog_name_seperator.
		For OLE DB use DBLITERAL_CATALOG_SEPARATOR or DBLITERAL_SCHEMA_SEPARATOR.
DataSourceProductName	string	The name of the product accessed by the provider, such as "Oracle" or "SQLServer".
DataSourceProductVersion	string	The version of the product accessed by the provider, in the data sources native format and not in Microsoft format.
		In some cases DataSourceProductVersion and DataSourceProductVersionNormalized are the same value. With OLE DB and ODBC, these are always be the same because they are mapped to the same function call in the underlying native API.
DataSourceProductVersionNormalized	string	A normalized version for the data source, such that it can be compared with **String.Compare()**. The format of this is consistent for all versions of the provider to prevent version 10 from sorting between version 1 and version 2.
		For example, the Oracle provider uses a format of "nn.nn.nn.nn.nn" for its normalized version, which causes an Oracle 8i data source to return "08.01.07.04.01". SQL Server uses the typical Microsoft "n.nn.nnnn" format.
		In some cases, DataSourceProductVersion and DataSourceProductVersionNormalized will be the same value. In the case of OLE DB and ODBC these will always be the same as they are mapped to the same function call in the underlying native API.
GroupByBehavior	GroupBy Behavior	Specifies the relationship between the columns in a GROUP BY clause and the non-aggregated columns in the select list.
IdentifierPattern	String	A regular expression that matches an identifier and has a match value of the identifier. For example "[A-Za-z0-9_#$]".
IdentifierCase	Identifier Case	Indicates whether non-quoted identifiers are treated as case sensitive.
OrderByColumnsInSelect	bool	Specifies whether columns in an ORDER BY clause must be in the select list. A value of true indicates that they are required to be in the select list, a value of false indicates that they are not required to be in the select list.

ParameterMarkerFormat	string	A format string that represents how to format a parameter.
		If named parameters are supported by the data source, the first placeholder in this string should be where the parameter name should be formatted.
		For example, if the data source expects parameters to be named and prefixed with an ':' this would be ":{0}". When formatting this with a parameter name of "p1" the resulting string is ":p1".
		If the data source expects parameters to be prefixed with the '@', but the names already include them, this would be '{0}', and the result of formatting a parameter named "@p1" would just be "@p1".
		For data sources that do not expect named parameters and expect the use of the '?' character, the format string can be specified as just '?', which would ignore the parameter name. For OLE DB we return '?'.
ParameterMarkerPattern	string	A regular expression that matches a parameter marker. It has a match value of the parameter name, if any.
		For example, if named parameters are supported with an '@' lead-in character that will be included in the parameter name, this would be: "(@[A-Za-z0-9_$#]*)".
		However, if named parameters are supported with a ':' as the lead-in character and it is not part of the parameter name, this would be: ":([A-Za-z0-9_$#]*)".
		Of course, if the data source does not support named parameters, this would just be "?".
ParameterNameMaxLength	int	The maximum length of a parameter name in characters. Visual Studio expects that if parameter names are supported, the minimum value for the maximum length is 30 characters.
		If the data source does not support named parameters, this property returns zero.
ParameterNamePattern	string	A regular expression that matches the valid parameter names. Different data sources have different rules regarding the characters that may be used for parameter names.
		Visual Studio expects that if parameter names are supported, the characters "\p{Lu}\p{Ll}\p{Lt}\p{Lm}\p{Lo}\p{Nl}\p{Nd}" are the minimum supported set of characters that are valid for parameter names.
QuotedIdentifierPattern	string	A regular expression that matches a quoted identifier and has a match value of the identifier itself without the quotes. For example, if the data source uses double-quotes to identify quoted identifiers, this would be: "(([^\"]|\"\")*)".
QuotedIdentifierCase	IdentifierCase	Indicates whether quoted identifiers are treated as case sensitive.
StatementSeparatorPattern	string	A regular expression that matches the statement separator.
StringLiteralPattern	string	A regular expression that matches a string literal and has a match value of the literal itself. For example, if the data source used single-quotes to identify strings, this would be: "('([^']|'')*')"'
SupportedJoinOperators	SupportedJoinOperators	Specifies what types of SQL join statements are supported by the data source.

DataTypes

Column Name	Data Type	Description
TypeName	string	The provider-specific data type name.
ProviderDbType	int	The provider-specific type value that should be used when specifying a parameter's type. For example, **SqlDbType.Money** or **OracleType.Blob**.
ColumnSize	long	The length of a non-numeric column or parameter refers to either the maximum or the length defined for this type by the provider.
		For character data, this is the maximum or defined length in units, defined by the data source. Oracle has the concept of specifying a length and then specifying the actual storage size for some character data types. This defines only the length in units for Oracle.
		For date-time data types, this is the length of the string representation (assuming the maximum allowed precision of the fractional seconds component).
		If the data type is numeric, this is the upper bound on the maximum precision of the data type.
CreateFormat	string	Format string that represents how to add this column to a data definition statement, such as CREATE TABLE. Each element in the **CreateParameter** array should be represented by a "parameter marker" in the format string.
		For example, the SQL data type DECIMAL needs a precision and a scale. In this case, the format string would be "DECIMAL({0},{1})".
CreateParameters	string	The creation parameters that must be specified when creating a column of this data type. Each creation parameter is listed in the string, separated by a comma in the order they are to be supplied.
		For example, the SQL data type DECIMAL needs a precision and a scale. In this case, the creation parameters should contain the string "precision, scale".
		In a text command to create a DECIMAL column with a precision of 10 and a scale of 2, the value of the CreateFormat column might be DECIMAL({0},{1})" and the complete type specification would be DECIMAL(10,2).
DataType	string	The name of the .NET Framework type of the data type.
IsAutoincrementable	bool	**true**—Values of this data type may be auto-incrementing.
		false—Values of this data type may not be auto-incrementing.
		Note that this merely indicates whether a column of this data type may be auto-incrementing, not that all columns of this type are auto-incrementing.
IsBestMatch	Bool	**true**—The data type is the best match between all data types in the data store and the .NET Framework data type indicated by the value in the DataType column.
		false—The data type is not the best match.

		For each set of rows in which the value of the DataType column is the same, the IsBestMatch column is set to true in only one row.
IsCaseSensitive	bool	**true**—The data type is a character type and is case sensitive. **false**—The data type is not a character type or is not case sensitive.
IsFixedLength	bool	**true**—Columns of this data type created by the data definition language (DDL) are of fixed length. **false**—Columns of this data type created by the DDL are of variable length. **DBNull.Value**—It is not known whether the provider will map this field with a fixed-length or variable-length column.
IsFixedPrecisionScale	bool	**true**—The data type has a fixed precision and scale. **false**—The data type does not have a fixed precision and scale.
IsLong	bool	**true**—The data type contains very long data; the definition of very long data is provider-specific. **false**—The data type does not contain very long data.
IsNullable	bool	**true**—The data type is nullable. **false**—The data type is not nullable. **DBNull.Value**—It is not known whether the data type is nullable.
IsSearchable	bool	**true**—The data type can be used in a WHERE clause with any operator except the LIKE predicate. **false**—The data type cannot be used in a WHERE clause with any operator except the LIKE predicate.
IsSearchableWithLike	bool	**true**—The data type can be used with the LIKE predicate. **false**—The data type cannot be used with the LIKE predicate.
IsUnsigned	bool	**true**—The data type is unsigned. **false**—The data type is signed. **DBNull.Value**—Not applicable to data type.
MaximumScale	short	If the type indicator is a numeric type, this is the maximum number of digits allowed to the right of the decimal point. Otherwise, this is **DBNull.Value**.
MinimumScale	short	If the type indicator is a numeric type, this is the minimum number of digits allowed to the right of the decimal point. Otherwise, this is **DBNull.Value**.
IsConcurrencyType	bool	**true**–The data type is updated by the database every time the row is changed and the value of the column is different from all previous values **false**–The data type is note updated by the database every time the row is changed **DBNull.Value**–The database does not support this data type.
IsLiteralsSupported	bool	**true**–The data type can be expressed as a literal. **false**–The data type cannot be expressed as a literal.
LiteralPrefix	string	The prefix applied to a given literal.

LitteralSuffix	string	The suffix applied to a given literal.
NativeDataType	String	An OLE DB-specific column for exposing the OLE DB type of the data type .

MetaDataCollections

Column Name	Data Type	Description
CollectionName	string	The name of the collection to pass to the **GetSchema** method to return the collection.
NumberOfRestriction	int	The number of restrictions that can be specified for the collection.
NumberOfIdentifierParts	int	The number of parts in the composite identifier/database object name. For example, in SQL Server, this would be 3 for tables and 4 for columns. In Oracle, it would be 2 for tables and 3 for columns.

ReservedWords

Column Name	Data Type	Description

Restrictions

Column Name	Data Type	Description
CollectionName	string	The name of the collection that these restrictions apply to.
RestrictionName	string	The name of the restriction in the collection.
RestrictionDefault	string	Ignored.
RestrictionNumber	int	The actual location in the collections restrictions that this particular restriction falls in.

Tables

Column Name	Data Type	Description
table_catalog	String	Catalog of the table.
table_schema	String	Schema that contains the table.
table_name	String	Table name.
table_type	String	Type of table. Can be VIEW or BASE TABLE.

Example

See Also

Other Resources

Obtaining Schema Information from the Host File System

BizTalk Adapter for Host Files

Managed Provider for Host Files Tutorial

This tutorial shows how to access host file data by using the Managed Provider for Host Files. After completing the tutorial, you will be able to do the following:

- Configure a connection to a host file dataset
- Build a simple XML Web service to expose host file data using ASP.NET
- Retrieve a recordset of information from a host file and return that information as a .NET dataset through an XML Web service.

Getting Started with the Managed Provider for Host Files Tutorial

Before you start the Managed Provider for Host Files tutorial, make sure to perform the following actions:

1. Install Microsoft® Visual Studio® 2005.
2. Ensure that Internet Information Services (IIS) is installed and ASP.NET 2.0 is configured.
3. Install Microsoft Host Integration Server 2009.
4. Ensure that you have access to a host dataset with permissions to query and modify data. For more information about the dataset used in this tutorial, see the Appendix.

Running the Managed Data Provider for Host Files Tutorial

To access host file data, you must first define the metadata associated with the dataset. Then you can use the Data Access Tool to configure a connection to the host file. After that, you can create the XML Web service and launch your application.

Step 1: Define the Structure of the Host File

To access the host dataset, you need to define the metadata associated with the dataset. After you have finished defining the various structures used to access the host file system, you can define the connection string you will use to connect to the file system.

To create the host file project

1. In Visual Studio, click **File**, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, in the **Project types** pane, click **Other Project Types**.
3. In the **Templates** pane, click **Blank Solution**.
4. In the **Name** field, enter **Data Integration Samples**, and then click **OK**.
5. In Solution Explorer, right-click **Data Integration Samples**, point to **Add**, and then click **New Project**.
6. In the **Add New Project** dialog box, in the **Project types** pane, click **Host Integration Projects**.
7. In the **Templates** pane, click **Host File Project**.
8. In the **Name** field, type **NorthwindHostFiles**, and then click **OK**.

To add a host file library to the host file project.

1. In Solution Explorer, right-click **NorthwindHostFiles**, point to **Add**, and then click **Add Host File Library**.
2. In the **Add New Item - NorthwindHostFiles** dialog box, in the **Name** field, type **NorthwindHostFiles_OS390**, and then click **Add**.
3. On the **Welcome to the Host Files Library Wizard** page, click **Next**.
4. On the **Host Environment** page, confirm that **Host Files for OS390** is selected in the **Host environment** drop-down

box, click **Next**, and then click **Create**.

To import the host file definition

1. In Solution Explorer, double-click **NorthwindHostFiles_OS390.DLL** to bring up the Host File Designer.
2. In the Host File Designer, right-click **NorthwindHostFiles_OS390**, point to **Import**, and then click **Host Definition**.
3. On the **Welcome to the COBOL Import Wizard** page, click **Next**.
4. On the **Import COBOL Source File** page, click **Browse** to locate the COBOL file that defines the data structures on your host file system, and then click **Next**.
5. On the **Structures member** page, select the COBOL group that represents the structure members, click **Next**, and then click **Modify**.

To map the host file schema to a table

1. In Host File Designer, expand **NorthwindHostFiles_OS390**, right-click **Tables**, and then click **Add Table**.
2. Right-click the newly created **table1**, and then click **Properties**.
3. In the Properties toolbox, enter an alias in the **Alias** field, the host file name in the **Host File Name** field, and a schema name in the **Schema** field.

To provide a strong key name to sign the host file project

1. In Host File Designer, right-click **NorthwindHostFiles_OS390**, and then click **Properties**.
2. In the **Properties** toolbox, expand **Assembly Information**, and then provide the path and file name to a key file.

To save and register the assembly

1. In Visual Studio, on the **File** menu, click **Save**.
2. Use the gacutil utility and the regasm tool to register the host file assembly.

Step 2: Create the Data Access String

After you have finished defining the structure of the host file, you can use the Data Access Tool to create a data access string. After you confirm the creation of the string, you can then create the XML Web service.

To start the Data Access Tool and run the Data Source Wizard

1. Click **Start**, point to **Programs**, point to **Microsoft Host Integration Server 2009**, and then click **Data Access Tool**.

   This starts the Data Access Tool, which you can use to configure connections in both DB2 and host file systems.
2. In the Data Access Tool, click **File**, point to **New**, and then click **Data Source**.
3. On the **Welcome to the Data Source Wizard** page, click **Next**.
4. On the **Data Source** page, in the **Data Source Platform** field, select **Mainframe or AS/36 file system**.
5. Confirm that the **SNA LU6.2 (APPC)** check box is selected, and then click **Next**.
6. On the **APPC Network Connection** page, enter the connection information for your mainframe, and then click **Next**.
7. On the **Mainframe or AS/36** page, enter the default information into the **Default library information** field.
8. Enter the path of the host file assembly created in the preceding section into the **Host File assembly** field, and then click **Next**.
9. On the **Locale** page, accept the default values, and then click **Next**.
10. On the **Security** page, in the **Security Method** field, confirm that **Interactive sign-on** is selected in the drop-down box.
11. Enter the connection information for your mainframe in the **Properties** fields, and then click **Next**.
12. On the **Advanced Options** page, click **Next**.
13. On the **Validation** page, click **Connect** to test your connection string, and then click **Next**.

14. On the **Saving Information** page, in the **Data source name** field, type **HIS_TRAINING_HF**.
15. Select the **Initialization string file** check box, and then click **Next**.
16. On the **Completing the Data Source Wizard** page, click **Finish**.

To view the data source string

1. Open Notepad and navigate to the directory the data source string is saved in.

   By default, the data source string file is located in C:\Documents and Settings\<USERNAME>\My Documents\Host Integration Projects\Data Sources.

2. Use Notepad to view the HIS_STRAINIGN_HF.txt file.

   You should see the connection string created by the Data Access Tool. It should look something like the following:

```
User ID=myname;Password=mypassword;APPC Remote LU Alias=DFM; APPC Local LU Alias=L3888888;APPC Mode
Name=PA62KNU; Network Transport Library=SNA; Host CCSID=37;PC Code Page=1252;Network Port=446;Default
Library=MyLibrary;Metadata='C:\Work\DataIntegrationSamples\NortwindHost
Files\NortwindHostFiles_OS390.DLL'
```

Step 3: Create the XML Web Service

After you have created the data string, you can create the XML Web service and associated code. Then you can test your code on a live Web site.

To start Visual Studio and create a new project

1. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2005**, and then click **Microsoft Visual Studio 2005**.
2. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, in the **Project types** pane, expand **Other Project Types**, and then click **Visual Studio Solutions**.
4. In the **Templates** pane, confirm that **Blank Solution** is selected, enter **HISTraining** in the **Name** field, and then click **Next**.

To add a class to hold the DB2 access logic

1. In Solution Explorer, right-click **HISTraining**, point to **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Project types** pane, expand **Other Languages**, and then click **Visual Basic**.
3. In the **Templates** pane, click **Class Library**.
4. In the **Name** field, type **HFDAL**, and then click **OK**.

To add a reference to the HostFileProvider

1. In Solution Explorer, right-click **HFDAL**, and then click **Add Reference**.
2. In the **Add Reference** dialog box, click the **.NET** tab, click the **Microsoft.HostIntegration.HostFileProvider** component, and then click **OK**.

To write the code for the Data Access class

1. In Solution Explorer, click **Class1.vb**.
2. Enter the following code into the text editor window:

```
Public Class HFDAL
    Public Function executeSQL(ByVal sqlQuery As String, ByVal connString As String) A
s DataSet
        Dim hfConn As New HostFileConnection(connString)
        Dim hfCmd As New HostFileCommand(sqlQuery, hfConn)
        Dim hfDA As New HostFileDataAdapter(hfCmd)
        Dim returnDS As New DataSet
```

```
        hfConn.Open()
        hfDA.Fill(returnDS)
        hfConn.Close()
        Return returnDS
    End Function

End Class
```

3. On the **File** menu, click **Save All**.
4. On the **Build** menu, click **Rebuild Solution**.

To create a Web service for the solution

1. In Solution Explorer, right-click **HISTraining**, point to **Add**, and then click **New Web Site**.
2. In the **Add New Web Site** dialog box, click **ASP.NET Web Service**.
3. In the **Location** combo box, confirm that **HTTP** is selected, type **http://localhost/HFWebService** in the associated text box, and then click **OK**.
4. In Solution Explorer, right-click **http://localhostHFWebService**, and then click **Add Reference**.
5. On the **Add Reference** dialog box, click the **Projects** tab, click **HFDAL**, and then click **OK**.
6. In Solution Explorer, expand **http://localhostHFWebService**, expand **App_Code**, and then double-click **Service.vb**.
7. Add the following code to Service.vb:

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function executeSQLQuery(ByVal connString As String, ByVal sqlStatement As
String) As Data.DataSet
        Dim dal As New HFDAL.HFDAL

        Return dal.executeSQL(sqlStatement, connString)

    End Function

End Class
```

8. On the **File** menu, click **Save All**.

Step 4: Launch the XML Web Service

After you are finished creating the XML Web service, you can launch the service and observe the resulting behavior.

To execute the Web service

1. In Solution Explorer, right-click **Service.asmx**, and then click **View in Browser**.
2. In the Web browser view of Service.asmx, click the hyperlink to **executeSQLQuery**.
3. On the **executeSQLQuery** page, copy the connection string from HIS_TRAINIGN_HF.TXT into the **connString** field.

4. In the **sqlStatement** field, type **SELECT* FROM CUSTOMERS**, and then click **Invoke**.

Appendix

The following COBOL copybook is used in this tutorial as the host file dataset.

```
01 CUSTOMER
     05 CUSTIDPIC S9(9)COMP.
     05 COMPANYPIC X(40).
     O5 CONTACT PIC X(30).
     05 TITLE PIC X(30).
     05 ADDRESS PIC X(60).
     05 CITY PIC X(20).
     05 REGION PIC X(15).
     05 ZIP PIC X(10).
     05 COUNTRY PIC X(10).
     05 PHONE PIC X(15).
     05 FAX PIC X(24).
     05 WEBPINPIC X(5).
```

See Also

Other Resources

Managed Data Provider for Host Files Programmer's Guide

.NET Framework Data Providers for Host Integration Server

In concept, a .NET Framework data provider is similar to an OLE DB provider: the provider runs as an interface between your application and the data source. However, the .NET Framework data provider exposes only a single set of ADO.NET interfaces.

In This Section

Examining the Core Interface for a Managed Provider

See Also

Other Resources

Managed Provider Programmer's Guide

Examining the Core Interface for a Managed Provider

A data provider in the .NET Framework serves as a bridge between an application and a data source. The data provider is used to retrieve data from a data source and to reconcile changes to that data back to the data source.

ADO.NET exposes a common model for .NET Framework data provider objects so that a single set of code can be written to work regardless of the .NET Framework data provider. The **Connection**, **Command**, **DataReader**, and **DataAdapter** objects represent the core elements of the .NET Framework data provider model. The following table describes the purpose of these objects, and how they are implemented in the Managed Provider for DB2 and Managed Provider for Host Files.

Common model	**Managed Provider for Host Integration Server**	**Description**
Connection	MsDb2Connection HostFileConnection	Responsible for opening, closing, and maintaining a connection to a DB2 host.
Command	MsDb2Command HostFileCommand	Manages all parameters that a query may include, which includes both SQL parameters and stored procedure parameters.
DataReader	MsDb2DataReader	A server-side forward-only cursor implementation that inherits form the **IDataReader** and **IDataRecord** interfaces.
DataAdapter	MsDb2DataAdapter HostFileDataAdapter	Acts as the gateway between the host data and a .NET Framework data set.

In addition, each provider has several interfaces specific to this implementation. These interfaces deal with exception and event handling, setting up connections to a DB2 host over different types of networks, and passing parameters.

See Also

Other Resources

.NET Framework Data Providers for Host Integration Server

Managed Provider Programmer's Guide

ADO.NET DataSet for Host Integration Server

The ADO.NET **DataSet** is the object that contains the data that a managed provider retrieves from a remote data source. Specifically, the ADO.NET **DataSet** is a local cache of tables, similar to a collection of ADO disconnected recordsets, described in the **System.Data** namespace. In addition, the **DataSet** records the relationships between the tables using keys and constraints. The **DataSet** is central to supporting disconnected, distributed data scenarios with ADO.NET.

In This Section

DataTableCollection

DataRelationCollection

ExtendedProperties

XML Support

DataTableCollection

An ADO.NET **DataSet** contains a collection of zero or more tables represented by **DataTable** objects. The **DataTableCollection** contains all the **DataTable** objects in a **DataSet**.

A **DataTable** is defined in the **System.Data** namespace and represents a single table of memory-resident data. It contains a collection of columns represented by a **DataColumnCollection**, and constraints represented by a **ConstraintCollection**, which together define the schema of the table. A **DataTable** object also contains a collection of rows represented by the **DataRowCollection**, which contains the data in the table. Along with its current state, a **DataRow** object retains both its current and original versions to identify changes to the values stored in the row.

See Also

Concepts

ADO.NET DataSet for Host Integration Server

Other Resources

Managed Provider Programmer's Guide

DataRelationCollection

A **DataSet** object contains relationships in its **DataRelationCollection** object. A relationship, represented by the **DataRelation** object, associates rows in one **DataTable** with rows in another **DataTable**. It is analogous to a join path that might exist between primary and foreign key columns in a relational database. A **DataRelation** identifies matching columns in two tables of a **DataSet**.

Relationships enable navigation from one table to another within a **DataSet**. The essential elements of a **DataRelation** are the name of the relationship, the name of the tables being related, and the related columns in each table. Relationships can be built with more than one column per table by specifying an array of **DataColumn** objects as the key columns. When a relationship is added to the **DataRelationCollection**, it may optionally add a **UniqueKeyConstraint** and a **ForeignKeyConstraint** to enforce integrity constraints when changes are made to related column values.

See Also
Concepts
ADO.NET DataSet for Host Integration Server
Other Resources
Managed Provider Programmer's Guide

ExtendedProperties

A **DataSet** object (like **DataTable** and **DataColumn**) has an **ExtendedProperties** property. **ExtendedProperties** is a **PropertyCollection** where you can add customized information, such as the SELECT statement that was used to generate the result set or a date/time stamp of when the data was generated. The **ExtendedProperties** collection is persisted with the schema information for the **DataSet** (and also the **DataTable** and **DataColumn**).

See Also

Concepts

ADO.NET DataSet for Host Integration Server

Other Resources

Managed Provider Programmer's Guide

XML Support

A **DataSet** object can persist and reload its contents as XML and its schema as XSD. The **DataSet** has direct support for reading (shredding) and writing data using XML using the .NET Framework **XmlReader** and **XmlWriter** classes. When you are writing XML, the output conforms to the W3C XSD schema. XML is a good method of moving data between application tiers. The **XMLDataDocument** can use the XML services, such as XSL/T and XPath.

See Also

Concepts

ADO.NET DataSet for Host Integration Server

Other Resources

Managed Provider Programmer's Guide

OLE DB Providers Programmer's Guides

This section of the Host Integration Server 2009 software development kit (SDK) provides information about using the OLE DB providers for AS/400, VSAM, and DB2.

For more information about OLE DB providers, see the Data Access section of the Operations guide.

This section contains:

- OLE DB Provider for AS/400 and VSAM Programmer's Guide
- OLE DB Provider for DB2 Programmer's Guide
- ADO Object, Method, Property, and Collection Support for AS/400, VSAM and DB2

OLE DB Provider for AS/400 and VSAM Programmer's Guide

The Microsoft® OLE DB Provider for AS/400 and VSAM enables you to directly access record-level data in mainframe VSAM, partitioned data sets (PDSs), and midrange OS/400 files from within an OLE-aware application. The object linking and embedding database (OLE DB) is a standard set of interfaces that provide heterogeneous access to disparate sources of information located anywhere—file systems, e-mail folders, and databases. The OLE DB Provider for AS/400 and VSAM combines the universal data access of OLE DB with the Record-Level Input/Output (RLIO) protocol of the IBM distributed data management (DDM) architecture.

DDM is a set of rules for distributing or extending the data management from one computer to another, such as from a mainframe to an AS/400 computer, or from one of these host computers to a server computer. By combining the OLE DB and DDM architectures, Microsoft enables organizations to preserve their investments in an existing data management infrastructure, while extending universal data access to all enterprise-wide data sources.

For API references and other technical information about the OLE DB provider, see the OLE DB Providers Programmer's Reference section of the SDK.

For more information about using the OLE DB Provider for AS/400 and VASM, see OLE DB Provider for AS/400 and VSAM in the Operations guide.

In This Section

See Also

Other Resources

Data Integration Programmer's Guide

Goals of the OLE DB Provider for AS/400 and VSAM

For the majority of enterprises today, much mission-critical information resides on IBM mainframe and AS/400 computers. This information is stored in records on the OS/400 and VSAM file systems. This information is created, owned, and often accessible by only host-based applications. In the mainframe environment, these applications include CICS and DB2; other commercial applications; and a large number of custom applications written in COBOL, PL/I, and other languages. In the AS/400 environment, these applications include primarily DB2 and commercial applications, plus a large number of custom report program generator (RPG) applications. Not all of these data sources are SQL-accessible. Many of the host data stores contain non-SQL-accessible data that is owned by something other than a traditional relational database management system (RDBMS).

These same enterprises rely on vast networks of personal computers to enable their users to achieve business goals. End users invariably rely on network e-mail, Microsoft Windows productivity applications such as Microsoft Office, and personal database programs such as Microsoft Access, to accomplish their daily tasks. It is essential for these same users to incorporate data stored on host systems into their regular correspondence, analysis, and reports.

Available methods of accessing host data do not provide the granular, record-level access required for cost-effective, more secure, and meaningful integration of host and personal computer systems. In many cases, end users employ antiquated means of data integration. These methods include copying and pasting data from a terminal emulation screen, retyping information from host application reports, and importing text files containing comma-delimited values that use host EBCDIC-to-computer ASCII file transfer. These methods are not efficient although widely used and are not supported by products from independent software vendors (ISVs).

The challenge is how to provide direct record-level access to this valuable data without going through the host application. Much of the renewed interest in improved access to host data sources is a result of the growth of local intranets, the use of the Internet, and Web technology as a mechanism for delivering information. Fast and inexpensive methods of record-level access are needed to deliver modern, three-tiered information systems during this era of cost-cutting and budget tightening. Additional uses of this direct data access are specific queries and Web-based reporting.

It is common for corporate management to rethink host data storage and the appropriate software used to provide data access. For many organizations, the answer to these issues is in rewriting the arguably outdated host-based business rules with server-based, or even client-based, business logic.

The goal of the Microsoft OLE DB Provider for AS/400 and VSAM is to provide customers and solution providers with the means to integrate desktop applications with this wealth of data residing on host computers.

OLE DB Environment

Three main roles are performed by software applications in an OLE DB environment:

- **OLE DB Consumer.** The end-user or server-based program that uses (consumes) the OLE DB interfaces. An example is a Web-based component that makes OLE DB calls to integrate host records with a Web-based report.

- **OLE DB Data Provider.** A driver or other program that exposes OLE DB interfaces for use by consumer applications. Data providers translate OLE DB interfaces to a language or commands that the target data source understands. An example is the Microsoft® OLE DB Provider for AS/400 and VSAM, which translates OLE DB interfaces to distributed data management (DDM) commands.

- **OLE DB Service Provider.** An application that both uses (consumes) and exposes OLE DB interfaces. Service providers typically act as proxies for the consumer, retrieving the data through the data provider and offering services to the consumer by manipulating the target data. An example is a query-processing engine.

DDM Record-Level Access

The Microsoft® OLE DB Provider for AS/400 and VSAM provides record-oriented access to host files. There is no need to perform bandwidth-intensive file transfers of entire host files to access data on the host.

The OLE DB interface provided by the OLE DB Provider for AS/400 and VSAM supports the following features:

- Set attributes and a record description of a host file (column information).
- Lock files and records.
- Position to the first record or the last record in a file.
- Navigate to the previous or next record in a file.
- Seek to a record, based on an index.
- Change records in a file.
- Insert new records and delete records in a file.
- Preserve file and record attributes.

The OLE DB Provider for AS/400 and VSAM is a source distributed data management (DDM) requester implementation that can initiate DDM commands to be serviced by a remote host-based target DDM server. On the Microsoft® Windows Server™ 2003 or Windows® 2000 operating system, the Microsoft DDM requester can run as a Windows service. This enables the DDM service to integrate with other host applications using the IBM DDM protocol and DDM servers that are resident on the host. Microsoft-based host software is not required. For more information, see Platforms Supported by the OLE DB Provider for AS/400 and VSAM. IBM offers DDM servers for the most popular host environments.

Providing users with direct record-level access reduces the development time to build and deploy new data integration solutions. Accessing only the target records, as opposed to entire host files, helps ensure data integrity.

Platforms Supported by the OLE DB Provider for AS/400 and VSAM

On the mainframe platform, IBM offers a target distributed data management (DDM) server implementation in IBM Distributed File Manager (DFM), a component of IBM Data Facility Storage Management Subsystem (DFSMS). The Microsoft OLE DB Provider for AS/400 and VSAM requires DFSMS version 1 release 2 or later for MVS/ESA and OS/390 to support an SNA LU 6.2 connection.

On midrange AS/400 computers, IBM has implemented target DDM servers directly in OS/400. The OLE DB Provider for AS/400 and VSAM requires OS/400 Version 3 Release 2 or later to support an SNA LU 6.2 connection. The OLE DB Provider for AS/400 and VSAM requires OS/400 Version 4 Release 2 or later to support a TCP/IP connection.

On the AS/400 platform, the OLE DB Provider for AS/400 and VSAM supports physical and logical files with an associated external record description file. For specific limitations, see the *AS/400 DDM User's Guide*.

On the mainframe platform, the OLE DB Provider for AS/400 and VSAM supports the following data set types:

Sequential Access Method (SAM) data sets

- Basic Sequential Access Method data sets (BSAM)
- Queued Sequential Access Method data sets (QSAM)

Virtual Storage Access Method (VSAM) data sets

- Entry-Sequenced Data Sets (ESDS)
- Key-Sequenced Data Sets (KSDS)
- Fixed-Length Relative Record Data Sets (RRDS)
- Variable-Length Relative Record Data Sets (VRRDS)
- Relative Record Data Set (RRDS)
- VSAM Alternate Indexes for ESDS and KSDS data sets

Basic Partitioned Access Method (PDS) data sets

- Partitioned Data Set Extended members (PDSE)
- Partitioned data set (PDS) members
- Read-only support for PDSE directories
- Read-only support for PDS directories

The preceding data set types are supported by IBM DFM/MVS.

The following data set types are not supported by DFM/MVS and cannot be accessed using the OLE DB Provider for AS/400 and VSAM:

- VSAM Linear Data Sets (LDS)
- Generation Data Groups (GDG)
- Generation Data Sets (GDS)

- Basic Direct Access Method data sets (BDAM)
- Indexed Sequential Access Method data sets (ISAM)
- Sequential Data Striping data sets
- OpenEdition MVS Hierarchical File System (HFS) files
- Tape Media

All mainframe data sets accessible through IBM Distributed File Manager must be cataloged in an Intersystem communications function (ICF) catalog and reside on direct access storage devices (DASD).

The OLE DB Provider for AS/400 and VSAM supplied with Host Integration Server 2009 supports the following OLE DB and ADO versions:

- **OLE DB version 2.5.** The Host Integration Server 2009 data access features require the run-time libraries for OLE DB version 2.5. On Windows Server 2003 or Windows 2000, these OLE DB libraries are installed as part of the operating system.
- **ADO version 2.5.** The Host Integration Server data access features require the run-time libraries for ADO version 2.5. On Windows Server 2003 or Windows 2000, these ADO libraries are installed as part of the operating system.

OLE DB version 2.0 or later and ADO version 2.0 or later are required to support indexed record access from an ADO consumer application using the OLE DB Provider for AS/400 and VSAM. Indexed support through OLE DB is supported with OLE DB versions 2.0 and later.

Indexed File Access

The Microsoft® OLE DB Provider for AS/400 and VSAM provides both sequential and indexed file access. Sequential file access is provided for all supported file types on the Platforms Supported by the OLE DB Provider for AS/400 and VSAM.

Indexed file access is provided for the following host file types only:

- Mainframe Virtual Storage Access Method (VSAM) data sets.
    - Key-Sequenced Data Sets (KSDS) only when the keys are unique.
    - Fixed-length Relative Record Data Sets (RRDS) only when the keys are unique.
    - Variable-length Relative Record Data Sets (VRRDS) only when the keys are unique.
- AS/400 files.
    - Logical files.
    - Keyed physical files (externally described to the system).

OLE DB and ADO offer several interfaces that enable indexed file access.

The OLE DB Provider for AS/400 and VSAM supports integrated indexes based on the underlying rowset. OLE DB support for indexed file access using the OLE DB Provider for AS/400 and VSAM is available using the **IRowsetIndex**, **IViewFilter**, and **IViewRowset** interfaces.

For more information about indexes, see Chapter 8, "Indexes" and Chapter 16, "Integrated Indexes" in the *OLE DB Programmer's Reference*. To obtain a list of available indexes in a target AS/400 library, a program can call the OLE DB **IDBSchemaRowset::GetRowset** function of the **Session** object requesting a query type of DBSCHEMA_INDEXES.

ADO support for indexed file access using the OLE DB Provider for AS/400 and VSAM is available using the Find method, Filter property, and Sort property on the ADO **Recordset** object. To obtain a list of available indexes in a target AS/400 library using ADO, a program can call the OpenSchema method on the **Connection** object specifying a *QueryType* of **adSchemaIndexes**.

By default, the OLE DB Provider for AS/400 and VSAM uses a server-based cursor. This means that all indexed file access is based on the cursor located over the host file and not a local computer copy of the host file. If you want to use the many client-based cursor service providers available with the Microsoft® Data Access Components (MDAC), you must configure the provider to use a client-based cursor. For example, a client-based cursor is required when using Remote Data Service (RDS) and the Microsoft® Visual Studio® ADO data-bound controls. However, using these controls, an application can access the host files for read-only purposes. If your application needs to access host files with an intent of both reading and writing, and you require indexed file access, your application should use the OLE DB Provider for AS/400 and VSAM server-based cursor.

File and Record Attributes

By definition, the record description is not part of the record I/O architecture in distributed data management (DDM). Traditionally, applications must embed the record format as part of the application program. This creates a burden on the application and is inconsistent with the existing computer-based data access standards, such as OLE DB and ODBC.

To solve this problem, the Microsoft® OLE DB Provider for AS/400 and VSAM uses an external host column description (HCD) file stored on the computer that enables administrators to describe the host record format. At run time, the OLE DB Provider for AS/400 and VSAM transparently converts the host data to computer data using the local HCD information. Before a user program can view or open a VSAM file using the OLE DB Provider for AS/400 and VSAM, the user program must create a valid record description file or entry for the target VSAM file.

The OLE DB Provider for AS/400 and VSAM includes a Microsoft Management Console (MMC) application designed to enable administrators and developers to easily create these local record description files and the necessary registry settings for data sources. The OLE DB DDM Management application makes it relatively easy to create HCD files without knowing the HCD file format. The Host Column Description file format is documented in the Data Integration Programmer's Reference.

The conversion process occurs in two steps. The host data is converted from host EBCDIC to ASCII data by the DDM dynamic-link library (DLL). The HCD file is used during this step to convert host data types to C data types, which are defined in ODBC and based on the SQL data types defined in the ANSI/ISO SQL-92 standard. The second phase of this conversion occurs in the SNAOLEDB DLL where these SQL C data types are converted to the defined OLE DB data types.

The use of an HCD file is not necessary to describe the record format for data stored in the AS/400 computer because the OLE DB Provider for AS/400 and VSAM automatically detects that the target host system is an AS/400 and uses the appropriate DDM commands to retrieve the record description. If the system administrator or the OLE DB application developer wants to use an HCD file instead of retrieving the AS/400 record description, this behavior can be forced by setting the configuration of the **Host Column Description File** property using Data Links.

Configuring the OLE DB Provider for AS/400 and VSAM

Microsoft® Data Access Components (MDAC) includes Data Links, a generic method for managing and loading connections to OLE DB data sources. Microsoft Data Links provide a uniform method of creating persistent OLE DB data source object definitions stored in the form of universal data link (.udl) files. The Microsoft® OLE DB Provider for AS/400 and VSAM normally uses Data Links and .udl files for loading and configuring data sources.

Data Links provide a flexible method for finding and saving connection information to OLE DB data sources.

The Data Source Wizard in the new Microsoft Data Access Tool can help to define .udl files. OLE DB consumer applications, such as Data Transformation Services in Microsoft® SQL Server™ can then use the .udl file to connect to IBM data sources, such as DB2 and the mainframe file system.

To use Microsoft OLE DB Provider for AS/400 and VSAM with an OLE DB consumer application, the user must either create a Microsoft data link (.udl) file and call this from the application, or call the OLE DB provider from within the application using a connection string that includes the provider name and other necessary parameters. If an application accesses VSAM data sets, after configuring a data link, a host data description must also be configured using the Data Descriptions tool.

This section contains:

Creating Data Links for the OLE DB Provider for AS/400 and VSAM

Data source information must be configured for each AS/400 or mainframe system data source object that is to be accessed using the OLE DB provider. The default parameters for the OLE DB provider are used as the default values for data sources when these parameters are not configured for each data source.

Microsoft Data Links, a core element of Microsoft Data Access Components (MDAC), provide a uniform method for creating file-persistent OLE DB data source object definitions in the form of universal data link (.udl) files.

The Data Source Wizard in the new Microsoft Data Access Tool can help to define .udl files. OLE DB consumer applications, such as Data Transformation Services in Microsoft SQL Server can then use the .udl file to connect to IBM data sources, such as DB2 and the mainframe file system.

Applications, such as the RowsetViewer sample included with the Microsoft Data Access SDK and the MSDN Platform SDK, can open created .udl files and pass the stored initialization string to the OLE DB Provider for AS/400 and VSAM at run time.

Creating New Data Links for the OLE DB Provider for AS/400 and VSAM

You create new Data Links using the Data Source Wizard from the Data Access Tool.

Browsing Data Links for the OLE DB Provider for AS/400 and VSAM

You can browse data sources using the Data Source Browser window in the Data Access Tool.

To find the physical location of the .udl file associated with a data source, right-click the data source and select **Locate** from the context menu. Windows Explorer appears which will display the location of the file.

Configuring Data Links for the OLE DB Provider for AS/400 and VSAM

To edit the properties of a Data Link file, browse to that file in Windows Explorer, and right-click the file and then click **Data Link Properties**. The **Data Link Properties** dialog box appears with several property tabs:

- **General**
- **Security**
- **Summary**
- **Provider**
- **Connection**
- **Advanced**
- **All**

The **General**, **Security**, and **Summary** tabs provide access to general file information for the .udl file that is available for other files and is not related to the Data Link properties. This information includes file location, file type, file size, file dates, file security permissions for access, and descriptive summary information (description and origin properties and values such as title, subject, and author) for the .udl file. The **General** tab has a text box with the name of the Data Link. This file name must end with the .udl extension if the file is to be recognized as a Data Link file. Note that the **Security** and **Summary** tabs are available on NTFS files systems, and not on the older FAT file systems.

This section contains:

Provider

The **Provider** tab enables you to select the OLE DB provider (the provider name string) to be used in this .udl file from a list of possible OLE DB providers. Select the Microsoft OLE DB Provider for AS/400 and VSAM. The parameters and fields displayed by the remaining tabs (**Connection**, **Advanced**, and **All**) are determined by the OLE DB provider that is selected.

Connection

The **Connection** tab enables you to configure the basic properties required to connect to a data source. For the Microsoft OLE DB Provider for AS/400 and VSAM, the connection properties include the following values.

Property	Description
Data Source	This is an optional parameter that can be used to describe the data source.
Network	This drop-down list box enables you to select the type of network connection to be used. The allowable options are **TCP/IP Connection** or **SNA Connection**. If **TCP/IP Connection** is selected, click **More Options**, to open a dialog box for configuring TCP/IP network settings. The parameters you can configure include the IP address of the remote host (or a hostname alias for this computer) and the Network Port (TCP/IP port) used for communication with the host. The default value for the Network Port is 446. The IP address of the host has no default value. If **SNA Connection** is selected (using LU 6.2), click **More Options** to open a dialog box for configuring SNA network settings. The parameters you can configure include: the APPC local LU alias, the APPC remote LU alias, and the APPC mode used for communication with the host. The default value for the APPC mode normally defaults to **QPCSUPP**. The local and remote LU alias fields do not have default values.
	This button enables using the Host Integration Security features providing a to access this OLE DB data source. When this button is selected, the **User name** and **Password** fields are dimmed and become inaccessible. The user name and password fields are set based on the Windows Server 2003 or Windows 2000 logon values. When this button is not selected, the **User name** and **Password** fields must normally contain appropriate values to access data sources on hosts.
User name	A valid user name and password are normally required to access data sources on hosts. These values are case-sensitive. The user must click the button that requires a specific user name and password to be entered.
Password	A valid user name and password are normally required to access data sources on hosts. These values are case-sensitive. The **Blank password** check box is only applicable for a Test Connection. To enter a password, the user needs to clear the **Blank password** check box if it is selected. If **Blank password** is selected, a Test Connection with a blank password does not cause the OLE DB provider to prompt for a password. Optionally, the user can choose to save the password in the .udl file by selecting the **Allow saving password** check box. Users and administrators should be warned that this option saves the authentication information (password) in plain text within the .udl file.
Location	The remote database name used for connecting to OS/400 systems. In DB2/400, this property is referred to as RDBNAM. This parameter is not used when connecting to mainframe systems.

It is possible to connect using a specific user name and password defined on the host system or use the feature (often referred to as Windows integrated security). If a specific user name and password is to be used, this information may need to be saved into the .udl file. The user name and password are saved in plain text in the .udl file. For security reasons in these cases, it is imperative that the .udl file be protected with an access control list (ACL) that restricts access to only authorized users. Saving the user name and password in the data link also forces this .udl file to be updated whenever the password associated with the username is changed. So for a variety of reasons, specifying a user name and password is not the preferred authentication option. Using the Windows integrated security option is the preferred method for authentication.

The **Connection** tab also includes a **Test Connection** button that can be used to test the connection parameters. The connection can only be tested after all of the required parameters are entered. When you click this button, an APPC session or a TCP/IP session attempts to be established to the host using the OLE DB Provider for AS/400 and VSAM.

Advanced

The **Advanced** tab enables you to set the network protection level and access permissions. You can set the protection level from the list box of allowable values. Access permissions are set by selecting the appropriate check boxes. For the Microsoft OLE DB Provider for AS/400 and VSAM, these properties include the values listed in the following table.

Property	Description
Host CCSID	The character code set identifier (CCSID) matching the column data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the Process Binary as Character value is set to true, character data is converted based on the column CCSID and default ANSI code page. This parameter defaults to **U.S./Canada (37)**. This parameter is equivalent to the DBPROP_SNAOLEDB_HOSTCCSID OLE DB property ID.
PC Code Page	The PC Code Page parameter indicates the code page to be used on the personal computer for character code conversion. This parameter is required when processing binary data as character data. Unless the Process Binary as Character check box is selected (value is set to true), character data is converted based on the default ANSI code page configured in Windows. This parameter defaults to **Latin 1 (1252)**. This parameter is equivalent to the DBPROP_SNAOLEDB_PCCODEPAGE OLE DB property ID.
Read only	When the Read-only parameter is selected in the **Advanced** tab, the OLE DB Provider for AS/400 and VSAM creates a read-only data source by setting the **Mode** parameter to **Read** (DB_MODE_READ). A user has read access to files and cannot do update operations.
Repair Host Keys	This parameter provides for repair of invalid key offsets received from OS/400 when keys have been defined using the DDS "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry. This parameter defaults to false. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.

All

The **All** tab allows users to configure essentially all of the properties for the data source except for the OLE DB provider. The properties available in the **All** tab include properties that can be configured using the **Connection** and **Advanced** tabs as well as optional detailed properties used to connect to a data source. Some of the properties in the **All** tab are required.

These properties on the **All** tab may be edited by selecting a property from the list displayed and selecting **Edit Value**. This button invokes a dialog box for the specific property containing a Property Description describing the property and a Property Value box for making changes.

For the Microsoft OLE DB Provider for AS/400 and VSAM, these properties include the following values.

Property	Description
Affiliate Application	This property is only used when is enabled. It provides the application name to use when retrieving host credentials from the database.
APPC Local LU Alias	The name of the local LU alias configured on the Host Integration Server 2009 computer. This parameter is equivalent to the DBPROP_SNAOLEDB_LOCALLU OLE DB property ID.
APPC Mode Name	When LU 6.2 (SNA) is selected for the Network Transport Library, this field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server computer configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to **QPCSUPP**. This parameter is equivalent to the DBPROP_SNAOLEDB_APPCMODE OLE DB property ID.
APPC Remote LU Alias	When LU 6.2 (SNA) is selected for the Network Transport Library, this field is the name of the remote LU alias configured in the Host Integration Server computer. This parameter is equivalent to the DBPROP_SNAOLEDB_REMOTELU OLE DB property ID.
Cache Authentication	This parameter determines whether the OLE DB provider caches authentication information such as a password in an internal cache. The value of this property (**true** or **false**) is selected from the drop-down list box. This parameter is not currently supported by the OLE DB provider and defaults to **false**. This parameter is equivalent to the DBPROP_AUTH_CACHE_AUTHINFO OLE DB property ID.
Connect Timeout	The amount of time (in seconds) to wait for initialization to complete. This parameter is not currently supported by the OLE DB provider and defaults to **0**. This parameter is equivalent to the DBPROP_INIT_TIMEOUT OLE DB property ID.
Data Source	The data source is an optional parameter that can be used to describe the data source. This parameter is equivalent to the DBPROP_INIT_DATASOURCE OLE DB property ID.

Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files. This parameter is equivalent to the DBPROP_SNAOLEDB_LIBRARY OLE DB property ID.
Encrypt Password	This parameter determines whether special security mechanisms are used to ensure password privacy. The value of this property (true or false) is selected from the drop-down list box. This parameter is not currently supported by the OLE DB provider and defaults to **false**. This parameter is equivalent to the DBPROP_AUTH_ENCRYPT_PASSWORD OLE DB property ID.
Extended Properties	This parameter is a string containing provider-specific, extended connection information. Properties passed through this parameter should be delimited by semicolons and will be interpreted by the provider's underlying network client. The use of this property implies that the OLE DB consumer knows how this string is interpreted and used by the OLE DB provider. This parameter should be used only for provider-specific connection information that cannot be explicitly described through the other property parameters. This parameter is equivalent to the DBPROP_INIT_PROVIDERSTRING OLE DB property ID.
Host CCSID	The character code set identifier (CCSID) matching the data as represented on the host. The CCSID property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the host column CCSID and default ANSI code page. This parameter defaults to **U.S./Canada (37)**. This parameter is equivalent to the DBPROP_SNAOLEDB_HOSTCCSID OLE DB property ID.
Host Column Description File	The fully qualified file name of the distributed data management (DDM) Host Column Description (HCD) file. This parameter can be a UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the system directory below where the Host Integration Server or Client software was installed. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400. This parameter is equivalent to the DBPROP_SNAOLEDB_HCDPATH OLE DB property ID.
Impersonation Level	The level of impersonation that the server is allowed to use when impersonating the client. This property applies only to network connections other than Remote Procedure Call (RPC) connections. These impersonation levels are similar to those provided by RPC. The values of this property correspond directly to the levels of impersonation that can be specified for authenticated RPC connections, but can be applied to connections other than authenticated RPC. This parameter can be set to one of the following values: **Anonymous**—The client is anonymous to the server. The server process cannot obtain identification information about the client and cannot impersonate the client. **Delegate**—The process can impersonate the client's security context while acting on behalf of the client. The server process can also make outgoing calls to other servers while acting on behalf of the client. **Identity**—The server can obtain the client's identity. The server can impersonate the client for access control list (ACL) checking, but cannot access system objects as the client. **Impersonate**—The server process can impersonate the client's security context while acting on behalf of the client. This information is obtained when the connection is established, and not on every call. The value of this property is selected from the drop-down list box. This parameter defaults to **Impersonate**. This parameter is equivalent to the DBPROP_INIT_IMPERSONATION_LEVEL OLE DB property ID.

Integrated Security	This parameter is a string containing the name of the authentication service used by the server to identify the user using the identity provided by an authentication domain. For example, for Microsoft Windows Server 2003 or Windows 2000 Integrated Security, this is "SSPI" (for Security Support Provider Interface). If this parameter is a null pointer, the default authentication service should be used. When this property is used, no other DBPROP_AUTH* properties are needed and, if provided, their values are ignored. This parameter is equivalent to the DBPROP_AUTH_INTEGRATED OLE DB property ID.
Locale Identifier	This parameter specifies the locale to be used. OLE DB Provider for AS/400 and VSAM does not support this parameter and defaults to **437**. This parameter is equivalent to the DBPROP_INIT_LCID OLE DB property ID.
Location	The remote database name used for connecting to OS/400 systems. In DB2/400, this property is referred to as RDBNAM. This parameter is not used when connecting to mainframe systems. This parameter is equivalent to the DBPROP_INIT_LOCATION OLE DB property ID.
Mask Password	This parameter indicates whether the password should be sent to the data source or enumerator in a masked form. The value of this property (**true** or **false**) is selected from the drop-down list box. The OLE DB provider does not support this parameter and defaults to **false**. This parameter is equivalent to the DBPROP_AUTH_MASK_PASSWORD OLE DB property ID.
Mode	After a connection is established, this parameter represents a bitmask of the access permissions that will be applied to the data file. As implemented by the OLE DB Provider for AS/400 and VSAM, access permissions apply to host file locks and do not apply to record locks. The allowable values include the following: Read, ReadWrite, Share Deny None, Share Deny Read, Share Deny Write, Share Exclusive, and Write. This parameter can be a combination of zero or more of the following: DB_MODE_READ—Read-only. DB_MODE_WRITE—Write-only. DB_MODE_READWRITE—Read/write (DB_MODE_READ | DB_MODE_WRITE). DB_MODE_SHARE_DENY_READ—Prevents others from opening in read mode. DB_MODE_SHARE_DENY_WRITE—Prevents others from opening in write mode. DB_MODE_SHARE_EXCLUSIVE—Prevents others from opening in read/write mode (DB_MODE_SHARE_DENY_READ | DB_MODE_SHARE_DENY_WRITE). DB_MODE_SHARE_DENY_NONE—Neither read nor write access can be denied to others. This parameter is equivalent to the DBPROP_INIT_MODE OLE DB property ID.
Network Address	When TCP/IP has been selected for the Network Transport Library, this parameter is used to locate the target host computer. This parameter indicates the IP address or TCP/IP host name alias associated with the DDM server on the host. The network address is required when connecting through TCP/IP. This parameter is equivalent to the DBPROP_SNAOLEDB_NETADDRESS OLE DB property ID.
Network Port	When TCP/IP has been selected for the Network Transport Library, this parameter is used to locate the target DDM service access port when connecting through TCP/IP. This parameter represents the TCP/IP port used for communication with the DDM service on the host. The default value is TCP/IP port 446. This parameter is equivalent to the DBPROP_SNAOLEDB_NETPORT OLE DB property ID.

Network Transport Library	This parameter, which represents the dynamic-link library used for transport, designates whether the provider connects through SNA LU 6.2 or TCP/IP for network communication. The possible values for this parameter are **TCPIP** or **SNA**. If TCPIP is selected, values for Network Address and Network Port are required. The OLE DB Provider for AS/400 and VSAM does not support TCP/IP connectivity to the mainframe. If SNA is selected, values for APPC Local LU Alias, APPC Mode Name, and APPC Remote LU Alias are required. The value of this property (SNA or TCPIP) is selected from the drop-down list box. This value defaults to **SNA**. This parameter is equivalent to the DBPROP_SNAOLEDB_NETTYPE OLE DB property ID.
Password	A valid user name and password are normally required to access data sources on hosts. The password is case-sensitive and is shown as asterisks in this dialog box for security purposes. Optionally, you can choose to save the password in the .udl file by selecting the **Allow saving password** check box. Users and administrators should be warned that this option persists the authentication information in plain text within the .udl file. This parameter is equivalent to the DBPROP_AUTH_PASSWORD OLE DB property ID.
PC Code Page	Indicates the code page used for character code conversion. This property is required when processing binary data as character data. Unless the **Process Binary as Character** value is set, character data is converted based on the default ANSI code page configured in the Windows operating system. If this parameter is set to **Binary** or **65535**, no character code conversions takes place. This parameter defaults to **Latin 1 (1252)**. This parameter is equivalent to the DBPROP_SNAOLEDB_PCCODEPAGE OLE DB property ID.
Persist Security Info	This parameter indicates whether the data source object is allowed to persist sensitive authentication information such as a password along with other authentication information. Optionally, a user can choose to save the password in the .udl file by selecting the **Allow saving password** check box. Users and administrators should be warned that this option persists the authentication information in plain text within the .udl file. The value of this property (**true** or **false**) is selected from the drop-down list box. This parameter defaults to **false**. This parameter is equivalent to the DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO OLE DB property ID.
Process Binary as Character	This parameter indicates whether to process binary fields (CCSID of 65535) as character data type fields on a per data source basis. The **Host CCSID** and **PC Code Page** values are required input parameters when this parameter is true. The value of this property (true or false) is selected from the drop-down list box. The default for this parameter is **false**; do not process binary fields as character fields. This parameter is equivalent to the DBPROP_SNAOLEDB_BINASCHAR OLE DB property ID.

Protection Level	The level of protection of data sent between client and server. The values of this property correspond directly to the levels of protection that can be specified for authenticated RPC connections. This parameter can be set to one of the following values: DB_PROT_LEVEL_NONE—Performs no authentication of data sent to the server. DB_PROT_LEVEL_CONNECT—Authenticates only when the client establishes the connection with the server. DB_PROT_LEVEL_CALL—Authenticates the source of the data at the beginning of each request from the client to the server. DB_PROT_LEVEL_PKT—Authenticates that all data received is from the client. DB_PROT_LEVEL_PKT_INTEGRITY—Authenticates all data received is from the client and that it has not been changed in transit. DB_PROT_LEVEL_PKT_PRIVACY—Authenticates all data received is from the client, that it has not been changed in transit, and protects the privacy of the data by encrypting it. The value of this property is selected from the drop-down list box. This parameter is not supported by the OLE DB Provider for AS/400 and VSAM and defaults to the connect level of protection. This parameter is equivalent to the DBPROP_INIT_PROTECTION_LEVEL OLE DB property ID.
Read only	When the Read only parameter is selected in the **Advanced** tab, the OLE DB Provider for AS/400 and VSAM creates a read-only data source by setting the **Mode** parameter to **Read** (DB_MODE_READ). A user has read access to files and cannot do update operations.
Repair Host Keys	This parameter provides for repair of invalid key offsets received from OS/400 when keys have been defined using the DDS "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry. This parameter defaults to **false**. The value of this property (**true** or **false**) is selected from the drop-down list box. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.
Strict Validation	This parameter indicates whether strict validation should be used and defaults to **false**. The value of this property (**true** or **false**) is selected from the drop-down list box. This parameter is equivalent to the DBPROP_SNAOLEDB_STRICTVAL OLE DB property ID.
User ID	A valid user name is normally required to access data sources on hosts. This value is case-sensitive. This parameter is equivalent to the DBPROP_AUTH_USERID OLE DB property ID.

Configuring Data Descriptions

You use the Data Access Tool to describe the host data file format for mainframe access. Use of the host column description (HCD) files is not required for AS/400 files because by default, the data file format is retrieved automatically from the host.

The data descriptions are stored in a local Host Column Description (HCD) file for each data source.

The following table provides the general parameters or attributes that can be configured describing each column in a data description on the **General** property page.

General parameters	Description
Name	The character string that represents the name of the column. This parameter may be null.
Alias	An optional character string that represents an alias label for the column string name. This parameter may be null.
Comment	An optional character string that represents a comment about the column. This attribute may be null.

The following table provides the data type parameters or attributes that can be configured describing each column in a data description in the **Host** property page.

Host parameters	Description
Type	The data type on the host. The allowed data values for host data type are selected from a drop-down list box. (For allowed values, see Host Data Types.)
Length	Length of the field in bytes. Depending on the selected host type, this parameter may not be editable.
Precision	Total number of decimal digits in the column containing numeric data on the host. Depending on the selected host type, this parameter may not be editable. The only numeric data types that require this information are the PACKED and ZONED data types. For these types, this field cannot be null. It must contain a valid numeric value. For all other host types, this parameter is not editable.
Scale	Number of decimal digits to the right of any decimal point for numeric data on the host. Depending on the selected host type, this parameter may not be editable. The only numeric data types that require this information are the PACKED and ZONED host data types. For these types, this field cannot be null. It must contain a valid numeric value. For all other host types, this parameter is not editable.
CCSID	The character code set identifier (CCSID) used on the host. The allowed data values for host CCSID are selected from a drop-down list box. This parameter defaults to the host CCSID configured for the OLE DB provider and is typically U.S./Canada (37).

The following table describes the parameters contained in the **Local** property page.

Local parameters	Description
Type	Indicates the OLE DB data type on the local computer. The allowed data values for the local data type are selected from a drop-down list box. (For allowed values, see Local OLE DB Data Types.)

The OLE DB provider limits the maximum length character field that can be accessed on an AS/400 computer to 32,745. On mainframes, a limitation of the IBM DFM is that SAM data sets and PDSE members are inaccessible if the fixed record length is greater than 32,760 or variable record lengths are greater than 32,756. DFM also limits all VSAM data sets on a mainframe to have a maximum record length no greater than 32,760. If you attempt to access a character field greater than these lengths on

an AS/400 or a mainframe, the attempt will fail and can have unpredictable results.

The CCSID setting used by the OLE DB provider must be set to match the CCSID actually used on the host—otherwise, data loss will occur. Some AS/400 systems default to a CCSID of 937 for enabling double-byte character sets (DBCS).

This section contains:

- Host Data Types
- Local OLE DB Data Types

Host Data Types

The **Host Type** parameter represents the data type used for this column on the host. The allowed values for **Host Type** that can be selected from the drop-down list box are listed in the following table.

Host Type	Description
Binary	Fixed-length binary data (no character conversion). The length must be specified for this data type.
Character	Fixed-length character data. The length must be specified for this data type.
Date	The date represented as character data in the format yyyy-mm-dd that fits into 10 bytes.
DBCS	Fixed-length character data that can contain only DBCS data. The length must be specified for this data type.
DCBS – Mixed Either	Fixed-length character data that can contain either DBCS or alphanumeric data. The length must be specified for this data type.
DCBS – Mixed Open	Fixed-length character data that can contain both DBCS and alphanumeric data. DBCS data is distinguished from alphanumeric data with SHIFT+CTRL characters. The length must be specified for this data type.
DBCS – Variable	Variable-length character data with a prefix of 2 bytes for length that contains only DBCS data. The maximum possible length for the column containing this data type must be specified.
DCBS – Variable Mixed Either	Variable-length character data with a prefix of 2 bytes for length that can contain either DBCS or alphanumeric data. The maximum possible length for the column containing this data type must be specified.
DCBS – Variable Mixed Open	Variable-length character data with a prefix of 2 bytes for length that can contain both DBCS and alphanumeric data. DBCS data is distinguished from alphanumeric data with SHIFT+CTRL characters. The maximum possible length for the column containing this data type must be specified.
Double	Floating-point data that fits in 8 bytes (64 bits).
Long	Integer data that fits in 4 bytes (32 bits).
Packed	Packed decimal numeric data where the precision and scale are exactly as specified.
Short	Integer data that fits in 2 bytes (16 bits).
Single	Floating-point data that fits in 4 bytes (32 bits).
Time	The time represented as character data in the format hh:mm:ss that fits into 8 bytes.
Time Stamp	Time stamp represented as characters in the format yyyy-mm-dd hh:mm:ss.ffffff that fits into 19 bytes.
Variable Binary	Variable-length binary data represented as an unsigned character array with a prefix of 2 bytes for length. The maximum possible length for the column containing this data type must be specified.
Variable Character	Variable-length character data with a prefix of 2 bytes for length. The maximum possible length for the column containing this data type must be specified.
Zoned	Zoned decimal numeric data where the precision and scale are exactly as specified.

Note

The floating-point data format assumed by the OLE DB Provider for AS/400 and VSAM depends on the host. For AS/400, the host floating-point data format is assumed to be IEEE. On mainframe hosts, floating-point data types are assumed to be in IBM floating-point formats. Because OLE DB supports the IEEE floating-point format, data conversion errors can occur when converting the extreme values of VSAM floating-point data in IBM format to IEEE floating-point data by the OLE DB provider. These conversion errors occur because the default IBM floating-point formats and the IEEE floating-point format use a different number of bits for the mantissa and exponent when representing a floating-point number.

Local OLE DB Data Types

The **Local Type** represents the OLE DB data type used for this column on the computer. The OLE DB data types are defined in the OLE DB specifications and **#defines** can be found in the Oledb.h file. The allowed values for **Local Type** that can be selected from the drop-down list box are listed in the following table.

Host Type	Description
DBTYPE_BYTES	Fixed-length binary data represented as an unsigned char array.
DBTYPE_DBDATE	The OLE DB DBDATE typedef struct as defined in the Oledb.h file.
DBTYPE_DBTIME	The OLE DB DBTIME typedef as defined in the Oledb.h file.
DBTYPE_DBTIMESTAMP	The OLE DB DBTIMESTAMP typedef struct as defined in the Oledb.h file.
DBTYPE_DECIMAL	The OLE DB DECIMAL typedef struct as defined in the Oledb.h file.
DBTYPE_I2	Integer data stored in 2 bytes (16 bits).
DBTYPE_I4	Integer data stored in 4 bytes (32 bits).
DBTYPE_NUMERIC	The OLE DB NUMERIC typedef struct as defined in the Oledb.h file.
DBTYPE_R4	Single precision IEEE floating-point data stored in 4 bytes (32 bits).
DBTYPE_R8	Double precision floating-point data stored in 8 bytes (64 bits).
DBTYPE_STR	Fixed and variable length character data.

Converting Existing Data Sources

The OLE DB management console that was previously used in SNA Server 4.0 and SNA Server 4.0 with Service Pack 1 (SP1) for configuring OLE DB Provider for AS/400 and VSAM data sources has been removed and replaced by Microsoft Data Links and the Data Access Tool. Microsoft Data Links is a component of Microsoft Data Access Components (MDAC) 2.5. On Windows Server 2003 or Windows 2000, MDAC 2.5 is installed as part of the operating system.

Existing registry-based OLE DB Provider for AS/400 and VSAM data sources that were created in SNA Server 4.0 and SNA Server 4.0 SP1 can be converted to .udl files using the Reg2udl tool supplied with Host Integration Server. The Reg2Udl tool is not installed as part of Host Integration Server, but is located on the Host Integration Server 2009 CD in the \Options\Maintenance folder.

When a duplicate .udl file is present in the destination folder, the Reg2udl tool will increment the file name by 1 (Data.udl will become Datat1.udl, for example). This may cause existing applications to fail because the OLE DB Provider for AS/400 and VSAM is looking for the existing name (Data.udl).

Manual conversion of registry-based data sources to .udl files may be necessary in some cases when Setup for Host Integration Server 2009 is used. A version of the Reg2udl tool (for Intel) can be found in the \Support\Utilities folder on the Host Integration Server 2009 CD.

Programming Considerations When Using the OLE DB Provider for AS/400 and VSAM

All the Microsoft® OLE DB objects exposed by the Microsoft® OLE DB Provider for AS/400 and VSAM support aggregation. Each OLE DB object has two classes, one that delegates its **IUnknown** calls and one that controls the object as a whole.

The apartment-threading model is supported, allowing multiple threads to access the objects safely. This is the only threading model supported.

When working with the Data Environment (DE) commands within Microsoft® Visual Studio® 6.0, you must use a period (.) as a delimiter when specifying the AS/400 Library or File path. For example, the following is valid syntax when opening the AUTHORS physical file in the PUBS library on an AS/400:

```
EXEC OPEN PUBS.AUTHORS
```

To use the ADO Recordset Findmethod or the ADO Filter property, an AS/400 logical file, an AS/400 keyed physical file, a mainframe KSDS file with a unique key, or a mainframe RRDS file with a unique key must be used. If this method is used on an AS/400 nonkeyed physical file or any other mainframe file type, this method fails.

When using RRDS files, the **Find** method fails when a search is executed using a column name. For example, the following Microsoft® Visual Basic® code will fail on an RRDS file with a column called Area:

```
RecordSet.Find "Area > '1111'", 0, adSearchForward, adBookmarkFirst
```

The error description indicates that a bookmark is invalid.

RRDS files do not have an index based on a column name and the value of the column data, so the syntax to the preceding ADO **Find** method call does not work for RRDS files. In a COBOL program designed to dynamically find a record in an RRDS file, the record position would be passed. For example, for the 75th record in the file, a COBOL program would pass a value of 75. The COBOL program would then use the returned record number and the record length to calculate the position of the first byte of the record in the file.

The SQL command parser of the Data Environment does not accept the forward slash (/) character. The OLE DB Provider for AS/400 and VSAM automatically substitutes a forward slash in place of the period and passes the correctly formatted path to your AS/400 computer.

When using the Data Environment with Microsoft Visual Basic 6.0, it is possible to get the following error when accessing the OLE DB Provider for AS/400 and VSAM:

```
"File is in use by another process. Unspecified error"
```

This error can occur after a data source has been configured for the OLE DB Provider for AS/400 and VSAM using the Data Links property page and a command is added using the Data Environment where the command added is the following:

```
"EXEC OPEN filename"
```

Using the Data Environment and selecting **Run** for this command can result in the preceding error. The Data Environment is opening the file and then trying to open it a second time based on the command to execute without closing the first copy. Depending upon the share options of the data set and the DBPROP_INIT_MODE property set for this data source, this error can occur and the user can be locked out of the AS/400 or VSAM file.

The OLE DB Provider for AS/400 and VSAM does not support the following Microsoft® SQL Server™ features:

- Replication
- Distributed queries as a linked server

However, the provider does support Data Transformation Services (DTS), with the following limitations:

1. Only bulk table copy (Import/Export) is available.

2. Only one data conversion layout (HCD map) per dataset.

   Specifically, there is no support for converting nested records, such as COBOL OCCURS or REDEFINE.

3. DTS can connect to DFM (a component of IMB SMS and Tivoli), but only over a SNA LU6.2 network connection.

4. Due to the architecture of the OLE DB provider, the SNAOLEDB connection is slower than Host File Transfer ActiveX control.

When operating on large VSAM files and only querying data on a subset of the records, using the **Filter** property is not desirable because of the performance impact. The entire VSAM file is transferred to the client for filtering. A better solution is to use the server cursor engine and the **Find** method.

The syntax supported by the OLE DB Provider for AS/400 and VSAM for command text is as follows:

```
EXEC COMMAND DDMCmd
```

where *DDMCmd* represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands allow you to request functions from the OS/400 operating system. Some examples are the DLTF (Delete File) or DSPFFD (Display File Description) commands. These are the same commands that could be issued on the command line if you were connected to an AS/400 using a 5250 terminal session. For a detailed list of possible commands, see the *OS/400 CL Reference* for your platform.

The syntax supported by the OLE DB Provider for AS/400 and VSAM to open a rowset (table) using command text is as follows:

```
EXEC OPEN FileName
```

where *FileName* represents one of the following host file naming conventions listed in the following table.

Host file type	File naming convention
VSAM Data Sets	DATASETNAME.FILENAME
Partitioned data sets (PDSs)	DATASETNAME.FILENAME(MEMBER)
OS/400 Files	LIBRARY/FILE
OS/400 Files	LIBRARY/FILENAME
OS/400 File Members	LIBRARY/FILE(MEMBER)
OS/400 File Members	LIBRARY.FILENAME(MEMBER)

Note that if a member of a library contains a dot in the member name, the member name must be surrounded by double quotes. For example, if the member name is NAMES.DAT, the proper syntax used to open a rowset using command text is as follows:

```
EXEC OPEN LIBRARY/FILE("NAMES.DAT")
```

The distributed queries feature of SQL Server is sometimes referred to as the Distributed Query Processor (DQP).

The AS/400 BIGINT data type is not supported in this release.

This section contains:

- Record Access and Data Conversion

- Record Locking
- Client Cursor Engines Using the OLE DB Provider for AS/400 and VSAM
- Error Codes Returned by the OLE DB Provider for AS/400 and VSAM

Record Access and Data Conversion

The design of the OLE DB APIs is similar to the APIs provided by ODBC and other ISAM APIs. The APIs are handle-based. After opening a file, the application can determine the buffer size required to store a row, use the cursor APIs to move, and optionally retrieve one or more rows of data using the row-level binding.

Data is converted to default C data types as defined in ODBC, as illustrated in the following table.

Host data type	Default C data type
Binary	unsigned char binary[]
Character	char string[]
Date (in character format)	date struct
Double	Double
Long	Int
Packed	unsigned char number[]
Short	Short
Single	Float
Time (in character format)	time struct
Time Stamp (in character format)	timestamp struct
Variable Binary	unsigned char binary[]
Variable Character	char string[]
Zoned	unsigned char number[]

Data conversions from a large numeric type to a small numeric type are supported (from DOUBLE to SINGLE and from INT to SMALLINT, for example). However, truncation and conversion errors can occur that will not be reported by the OLE DB Provider for AS/400 and VSAM.

The OLE DB Provider for AS/400 and VSAM has a number of other limitations:

- Positive signed floating-point values cannot be read from ZONED DECIMAL fields.
- No floating point values can be inserted into ZONED DECIMAL fields.
- No values can be inserted into single-precision FLOATING POINT fields.
- Positive signed floating-point values cannot be inserted into PACKED DECIMAL fields.
- The ADO **Find** method fails to locate the first record when the key is multiple columns and the first column is a VARCHAR or TIME data type.

You can use the OLE DB Provider for AS/400 and VSAM to access a System/36 computer. However, the automatic mapping from host to client does not occur correctly when you attempt to access the System/36 host. Instead, you must manually map the data types using a host column description (HCD) file. The following table indicates the data types you must map to when accessing a System/36 computer.

Host data type	OLE DB data type	Comments
Character	DBTYPE_STR	Null-terminated ASCII character string.

Zoned	DBTYPE_STR	The NUMERIC typedef structure defined in OLEDB.H. This is an exact numeric value with a fixed precision and fixed scale.

Record Locking

Distributed data management (DDM) supports record locks so that a requester can perform intended operations on a record without interference from concurrent users. Record locks are used only when the requester opens a file with intent to update the file and specifies that the file is to be shared with another user. Two types of record locks are supported. Record locks are handled automatically by the Microsoft OLE DB Provider for AS/400 and VSAM whenever users call **IRowsetChange::SetData** (in immediate mode) or **IRowsetUpdate** (in the delayed mode). The OLE DB Provider for AS/400 and VSAM locks the record, updates the record, and then releases the lock.

Client Cursor Engines Using the OLE DB Provider for AS/400 and VSAM

The Microsoft Data Access Components (MDAC) supports the option of a client cursor engine. This feature is implemented as part of OLE DB, ADO, and Remote Data Services (RDS). When using ADO, a client cursor is enabled by setting the **CursorLocation** property on the recordset to **adUseClient**.

The OLE DB Provider for AS/400 and VSAM does not support any updating capabilities when used with a client cursor engine. If a client cursor engine is enabled using RDS or ADO, the OLE DB provider cannot be used to update data on the host. The ADO recordset is treated as if it were read-only.

Error Codes Returned by the OLE DB Provider for AS/400 and VSAM

The Microsoft OLE DB Provider for AS/400 and VSAM supports ranges of error codes, as listed in the following table.

Error code range	Source	Definition
1–100	Ddmapi.dll	OLE DB error codes (see OLE DB Provider for AS/400 and VSAM).
256–511	Ddm.dll	IBM DDM documentation.
512–higher	Ddmwappc.dll	Errors specific to the OLE DB Provider for AS/400 and VSAM.

An invalid local LU alias at connect time yields "Network Error" instead of "Invalid Local LU Alias" error. No error is reported when connecting using a nonexistent default library value.

Host Column Description

The Microsoft® OLE DB Provider for AS/400 and VSAM uses a host column description (HCD) file to specify how data on the host is converted by the OLE DB provider. These HCD files are not necessary when used with IBM AS/400 computers because the host data format is automatically determined by the OLE DB provider. When used with AS/400 computers, the OLE DB provider uses default conversions from host data type to OLE DB data types. However, HCD files can be used with AS/400 computers to override the host data format and specify a particular local OLE DB data type to which the data is to be converted.

The following topics describe the host column description file format in detail and provide an example of an HCD file for illustration.

This section contains:

Host Column Description File Format

The Microsoft® OLE DB Provider for AS/400 and VSAM uses a host column description (HCD) file to describe the format of data files on IBM mainframes and dictate how the data in columns or fields in these files is to be converted by the OLE DB provider. Host column description files can also be used for data on AS/400 computers to override the data format description maintained by the AS/400 host.

The default naming convention for the HCD file is <*data source*>.hcd, where <*data source*> is the remote LU alias, and .hcd is the file extension of the record description file. For example, if the remote LU alias configured on Host Integration Server is named ABC01234, the host column description file would be named ABC01234.hcd as the default. Any name may be used for an HCD file as long as the file extension is .hcd.

All the Virtual Storage Access Method (VSAM) files that have local record descriptions are listed in the [Files] section of the record description file. Each file that is listed in the [Files] section has a record description section, and that section name is the same as the file name. For example, a VSAM file named PUBS/AUTHORS has its record description saved in the [PUBS/AUTHORS] section.

The first key of a record description section is the number of the columns for the file. The syntax is as follows:

```
numcol=<number of columns>
```

Where <*number of columns*> is the actual number of columns described in the section. Each column of a file is described by one key. The naming convention for the key name is as follows:

```
col<column number>
```

For example the ninth columns key is **col9**. Each key has an attribute list that contains eleven concatenated attributes that describe the column. Attributes are separated by semicolons.

Host Column Description Attributes

Attributes are shown in the following table.

Attribute	Field number	Description
Reserved	1	Always zero
Column Name	2	Character string
Alias	3	Character string
Precision	4	Numeric value
Scale	5	Numeric value
Length	6	Numeric value
Host Type	7	Keyword representing the host data type
OLE DB Type	8	Keyword representing the local OLE DB data type
Nullable	9	Y or N
CCSID	10	Numeric value
Title	11	Character string

Column Name

The Column Name attribute is the character string that represents the name of the column. This attribute may be null.

Alias

The Alias attribute is an optional character string that represents an alias label for the column string name. This attribute may be null.

Precision

The Precision attribute is the total number of decimal digits in the column containing numeric data on the host. The only two fixed-point numeric data types that require this information are the NUMERIC and DECIMAL keyword data types, and for these types this field cannot be null and must contain a valid numeric value.

The precision must be set the same as the length attribute for CHAR and BINARY keyword data types, and set to zero for the other types. (Note that under ODBC, the precision was also used to indicate the length of nonnumeric data types including character, date, time, and binary data types.) Precision has no default value and must not be left null.

Scale

The Scale attribute is the number of decimal digits to the right of any decimal point for numeric data on the host. The only two numeric data types that require this information are the NUMERIC and DECIMAL keyword data types, and for these types this field cannot be null and must contain a valid numeric value. For other numeric (the SINGLE and DOUBLE keywords, for example) and nonnumeric data types (binary, character, date, time, and timestamp), the scale should be set to zero. This field must not be left null and must contain a numeric value.

Length

The Length attribute is the total length of the data on the host. This field must not be left empty and must contain a numeric value.

Host Type

The Host Type attribute is a keyword value that represents the data type of the host data. This keyword value is based on standard data types used on AS/400 and VSAM files. If no keyword is entered, this attribute defaults to the BINARY keyword. The following table describes the allowable types for AS/400 and VSAM.

Host type name	Keyword	Comment
Binary	BINARY	Fixed-length binary data (no character code conversions). The length must be specified.

Character	CHAR	Fixed-length character data. The length must be specified.
Date	DATE	The date represented as character data in the format yyyy-mm-dd that fits in 10 bytes.
DBCS	DBCS	Fixed-length character data that can contain only DBCS data.
DCBS – Mixed Either	MIXED_EITHER	Fixed-length character data that can contain either DBCS or alphanumeric data.
DCBS – Mixed Open	MIXED_OPEN	Fixed-length character data that can contain both DBCS and alphanumeric data. DBCS data is distinguished from alphanumeric data with SHIFT+CTRL characters.
DBCS – Variable	VARDBCS	Variable-length character data with a prefix of 2 bytes for length that contains only DBCS data. The maximum possible length for the column containing this data type must be specified.
DCBS – Variable Mixed Either	VARMIXED_EITHER	Variable-length character data with a prefix of 2 bytes for length that can contain either DBCS or alphanumeric data. The maximum possible length for the column containing this data type must be specified.
DCBS – Variable Mixed Open	VARMIXED_OPEN	Variable-length character data with a prefix of 2 bytes for length that can contain both DBCS and alphanumeric data. DBCS data is distinguished from alphanumeric data with SHIFT+CTRL characters. The maximum possible length for the column containing this data type must be specified.
Double	DOUBLE	Floating-point data that fits in 8 bytes (64 bits).
Long	LONG	Integer data that fits in 4 bytes (32 bits).
Long Variable Binary	LONGVARBINARY	Variable-length binary data represented as an unsigned character array with prefix 2 bytes for length. The maximum possible length for the column containing this data type must be specified.
Long Variable Character	LONGVARCHAR	Variable-length character data with a prefix of 2 bytes for the length. The maximum possible length for the column containing this data type must be specified.
Packed	PACKED	Packed decimal numeric data where the precision and scale are exactly as specified.
Short	SHORT	Integer data that fits in 2 bytes (16 bits).
Single	SINGLE	Floating-point data that fits in 4 bytes (32 bits).
Time	TIME	The time represented as character data in the format hh:mm:ss that fits in 8 bytes.
Time Stamp	TIMESTAMP	Timestamp represented as characters in the format yyyy-mm-dd hh:mm:ss.ffffff that fits in 19 bytes.
Variable Binary	VARBINARY	Variable-length binary data represented as an unsigned character array with prefix 2 bytes for length. The maximum possible length for the column containing this data type must be specified.
Variable Character	VARCHAR	Variable-length character data with a prefix of 2 bytes for length. The maximum possible length for the column containing this data type must be specified.
Zoned	ZONED	Zoned decimal numeric data where the precision and scale are exactly as specified.

Note that the OLE DB provider limits the maximum length character field that can be accessed on an AS/400 computer to 32,745. Attempting to access a character field greater than this length on an AS/400 computer can have unpredictable results and can cause the OLE DB provider to stop responding.

Note that the floating-point data format assumed by the OLE DB Provider for AS/400 and VSAM depends on the host. For AS/400, the host floating-point data format is assumed to be IEEE. On mainframe hosts, floating-point data types are assumed to be in IBM floating-point formats. Because OLE DB supports the IEEE floating-point format, data conversion errors can occur when converting the extreme values of VSAM floating-point data in IBM format to IEEE floating-point data by the OLE DB provider.

Note that the DECIMAL, FLOAT, INTEGER, NUMERIC, REAL, and SMALLINT keywords should not be used in HCD files and should be replaced with the newer keywords as follows:

- The DOUBLE keyword replaces the FLOAT keyword.
- The LONG keyword replaces the INTEGER keyword.
- The PACKED keyword replaces the NUMERIC keyword.
- The SHORT keyword replaces the SMALLINT keyword.
- The SINGLE replaces the REAL keyword.
- The ZONED keyword replaces the DECIMAL keyword.

The Data Descriptions management console snap-in provided with Host Integration Server does not work properly with HCD files containing these older keywords and will give unpredictable results.

OLE DB Type

The OLE DB Type attribute is a keyword that represents the data type of the local personal computer data. This keyword value is based on the standard OLE DB data types as defined in the OLEDB.H header file included with the OLE DB SDK version 1.5 and later. The data structures for the date, time, and timestamp C data types are defined as typedefs in the OLEDB.H header file included with the OLE DB SDK. Similar data types are also used by ODBC. If no keyword is entered, this attribute defaults to the BINARY keyword.

For the decimal and numeric host data types, the OLE DB Type attribute must be set to DBTYPE_STR. These two fixed-point numeric types are currently converted to character data strings by the DDM layer of the Microsoft OLE DB Provider for AS/400 and VSAM. If these host types are converted to any of the defined OLE DB numeric C types, numeric accuracy can be lost.

The allowable types are listed in the following table.

OLE DB type name	Keyword	Comment
DBTYPE_BYTES	BINARY	Fixed-length binary data represented as an unsigned char array.
DBTYPE_DBDATE	DATE	The OLE DB DBDATE typedef struct as defined in the OLEDB.H header file.
DBTYPE_DBTIME	TIME	The OLE DB DBTIME typedef as defined in the OLEDB.H header file.
DBTYPE_DBTIMESTAMP	TIMESTAMP	The OLE DB DBTIMESTAMP typedef struct as defined in the OLEDB.H header file.
DBTYPE_DECIMAL	DECIMAL	The OLE DB DECIMAL typedef struct as defined in the OLEDB.H header file.
DBTYPE_I2	SHORT	Integer data stored in 2 bytes (16 bits).
DBTYPE_I4	LONG	Integer data stored in 4 bytes (32 bits).
DBTYPE_NUMERIC	NUMERIC	The OLE DB NUMERIC typedef struct as defined in the OLEDB.H header file.
DBTYPE_R4	FLOAT	Floating-point data stored in 4 bytes (32 bits).
DBTYPE_R8	DOUBLE	Floating-point data stored in 8 bytes (64 bits).
DBTYPE_STR	CHAR	Character data.
DBTYPE_I1	Not applicable	Not supported (signed tiny integer stored in one byte, 8 bits).

DBTYPE_I8	Not applicable	Not supported (signed long integer stored in eight bytes, 64 bits).
DBTYPE_UI1	Not applicable	Not supported (unsigned tiny integer stored in one byte, 8 bits).
DBTYPE_UI2	Not applicable	Not supported (unsigned short integer stored in two bytes, 16 bits).
DBTYPE_UI4	Not applicable	Not supported (unsigned long integer stored in four bytes, 32 bits).
DBTYPE_UI8	Not applicable	Not supported (unsigned long integer stored in eight bytes, 64 bits).

Nullable

The Nullable attribute indicates whether this field can be a null value. Legal values for this field are Y or N. If this field is empty, the default value for nullable is N. The current version of the OLE DB Provider for AS/400 and VSAM does support nullable fields, so this value must be set to N.

CCSID

The character code set identifier (CCSID) attribute indicates the character set used on the host. If this field is empty, the default value for CCSID is set to EBCDIC US English (37). The CCSID setting used by the OLE DB provider must be set to match the CCSID actually used on the host, otherwise data loss will occur. Note that some AS/400 systems default to a CCSID of 937, rather than 37, for enabling double-byte character sets (DBCS).

If the CCSID is set to 0 or 65535, no character translation will take place when converting character data from the host to the personal computer by the OLE DB provider. The allowable values for CCSID when used with OLE DB Provider for AS/400 and VSAM are listed in the following table.

EBCDIC character set	CCSID value
U.S./Canada	37
Germany	273
Denmark/Norway	277
Finland/Sweden	278
Italy	280
Latin America/Spain	284
United Kingdom	285
France	297
International	500

Note that this value needs to correspond to the CCSID used on the host.

Title

The Title attribute is an optional character string that represents a comment describing the column. This attribute may be null.

Host Column Description Example File

The following is an example for a host column description (HCD) file containing two sample database file descriptions—SAMPLE/ACCOUNTS and PUBS/AUTHORS. The first sample file (SAMPLE/ACCOUNTS) has four columns of data while the second example (PUBS/AUTHORS) has nine columns that must be described.

```
[files]
SAMPLE/ACCOUNTS=1
PUBS/AUTHORS=1

[SAMPLE/ACCOUNTS]
numcol=4
col1=0;CUST_NO;CUST_NO;8;0;0;ZONED;LONG;N;37;;
col2=0;CUST_NAME;CUST_NAME;0;0;40;CHAR;CHAR;N;37;;
col3=0;BALANCE;BALANCE;10;2;0;ZONED;FLOAT;N;37;;
col4=0;LAST_ACC;LAST_ACC;0;0;26;TIMESTAMP;TIMESTAMP;N;37;;

[PUBS/AUTHORS]
NumCol=9
Col1=0;AU_ID;AU_ID;11;0;11;CHAR;CHAR;N;37;;
Col2=0;AU_LNAME;AU_LNAME;0;0;40;VARCHAR;CHAR;N;37;;
Col3=0;AU_FNAME;AU_FNAME;0;0;20;VARCHAR;CHAR;N;37;;
Col4=0;PHONE;PHONE;0;0;12;CHAR;CHAR;N;37;;
Col5=0;ADDRESS;ADDRESS;0;0;40;VARCHAR;CHAR;N;37;;
Col6=0;CITY;CITY;0;0;20;VARCHAR;CHAR;N;37;;
Col7=0;STATE;STATE;2;0;2;CHAR;CHAR;N;37;;
Col8=0;ZIP;ZIP;0;0;5;CHAR;SHORT;N;37;;
Col9=0;CONTRACT;CONTRACT;0;0;9;BINARY;BINARY;N;37;;
```

Conversion from Host to OLE DB Data Types

The external record description for data files residing on mainframes is configured in a host column description (HCD) file using the SNA OLE DB management snap-in. This HCD file is used to convert from host EBCDIC data types to ASCII C data types for the computer in the Distributed Data Management (DDM) dynamic-link library (DLL). These C data types are then mapped to OLE DB data types by the SNAOLEDB DLL.

This section contains:

Default OLE DB Data Types

The following table indicates the default OLE DB data types that result from the mapping of the host data types by the Microsoft OLE DB Provider for AS/400 and VSAM.

Host data type	OLE DB data type	Comments
Binary	DBTYPE_BYTES	A fixed-length array of bytes (unsigned char).
Character	DBTYPE_STR	Null-terminated ASCII character string.
Date	DBTYPE_DBDATE	The DBDATE typedef struct defined in OLEDB.H header file.
DBCS	DBTYPE_STR	Null-terminated ASCII character string.
DCBS – Mixed Either	DBTYPE_STR	Null-terminated ASCII character string.
DCBS – Mixed Open	DBTYPE_STR	Null-terminated ASCII character string.
DBCS – Variable	DBTYPE_STR	Null-terminated ASCII character string.
DCBS – Variable Mixed Either	DBTYPE_STR	Null-terminated ASCII character string.
DCBS – Variable Mixed Open	DBTYPE_STR	Null-terminated ASCII character string.
Double	DBTYPE_R8	8-byte floating-point data.
Float	DBTYPE_R8	8-byte floating-point data.
Long Integer	DBTYPE_I4	4-byte signed integer.
Long Variable Binary	DBTYPE_BYTES	A fixed-length array of bytes (unsigned char).
Long Variable Character	DBTYPE_STR	Null-terminated ASCII character string.
Packed	DBTYPE_DECIMAL	The DECIMAL structure typedef defined in OLEDB.H. This is an exact numeric value with a fixed precision and fixed scale.
Real	DBTYPE_R4	4-byte floating-point data.
Short	DBTYPE_I2	2-byte signed integer.
Single	DBTYPE_R4	4-byte floating-point data.
Time	DBTYPE_DBTIME	The DBTIME typedef defined in OLEDB.H header file.
Time Stamp	DBTYPE_DBTIMESTAMP	The DBTIMESTAMP typedef defined in OLEDB.H header file.
Variable Binary	DBTYPE_BYTES	A fixed-length array of bytes (unsigned char).
Variable Character	DBTYPE_STR	Null-terminated ASCII character string.
Zoned	DBTYPE_NUMERIC	The NUMERIC typedef structure defined in OLEDB.H. This is an exact numeric value with a fixed precision and fixed scale.

The host Binary, VarBinary, and Long VarBinary data types are converted to SQL_C_CHAR type by the DDM DLL and mapped to the DBTYPE_BYTES data type by the SNAOLEDB DLL. The host Zoned data type is converted to SQL_C_CHAR type by the DDM DLL and mapped to the DBTYPE_NUMERIC data type by the SNAOLEDB DLL. The host Packed data type is converted to SQL_C_CHAR type by the DDM DLL and mapped to the DBTYPE_DECIMAL data type by the SNAOLEDB DLL.

DBDATE

The **DBDATE** structure typedef is defined as follows:

Syntax

```
typedef struct tagDBDATE {
    SHORT  year;
    USHORT month;
    USHORT day
} DBDATE;
```

Members

year

The year (0 to 9999) is measured from 0 A.D.

month

The month ranges from 1 to 12 representing January through December.

day

The day ranges from 1 to a maximum of 31, depending on the number of days in the month.

DBTIME

The **DBTIME** structure typedef is defined as follows:

Syntax

```
typedef struct tagDBTIME {
    USHORT hour;
    USHORT minute;
    USHORT second
} DBTIME;
```

Members

hour

The hour ranges from 0 to 23.

minute

The minute ranges from 0 to 59.

second

The second ranges from 0 to 59.

DBTIMESTAMP

The **DBTIMESTAMP** structure typedef is defined as follows:

Syntax

```
typedef struct tagDBTIMESTAMP {
    SHORT  year;
    USHORT month;
    USHORT day;
    USHORT hour;
    USHORT minute;
    USHORT second;
    ULONG  fraction
} DBTIMESTAMP;
```

Members

year

The year (0 to 9999) is measured from 0 A.D.

month

The month ranges from 1 to 12 representing January through December.

day

The day ranges from 1 to a maximum of 31, depending on the number of days in the month.

hour

The hour ranges from 0 to 23.

minute

The minute ranges from 0 to 59.

second

The second ranges from 0 to 59.

fraction

The fraction represents billionths of a second ranging from 0 to 999,999,999.

DECIMAL

The **DECIMAL** typedef structure is an exact numeric value with a fixed precision and fixed scale, stored in the same way as in OLE Automation. The **DECIMAL** typedef structure is defined as follows:

Syntax

```
typedef struct tagDECIMAL {
    USHORT wReserved;
    union {
        struct {
            BYTE scale;
            BYTE sign;
         };
        USHORT signscale;
    };
    ULONG Hi32;
    union {
        struct {
             ULONG Lo32;
             ULONG Mid32;
          };
         ULONGLONG Lo64;
    };
} DECIMAL;
```

Members

wReserved

This member is reserved and should be 0.

scale

Specifies the number of digits to the right of the decimal point and ranges from 0 to 28.

sign

The sign is 0 if positive, 0x80 if negative.

Hi32

The high part of the integer (32-bit aligned).

Mid32

The middle part of the integer (32-bit aligned).

Lo32

The low part of the integer (32-bit aligned).

For example, to specify the number 12.345, the scale is 3, the sign is 0, and the number stored in the 12-byte integer is 12345.

NUMERIC

The **NUMERIC** typedef structure is an exact numeric value with a fixed precision and fixed scale. The **NUMERIC** typedef structure is defined as follows:

Syntax

```
typedef struct tagNUMERIC {
    BYTE precision;
    BYTE scale;
    BYTE sign;
    BYTE val[16];
} DB_NUMERIC;
```

Members

precision

The maximum number of digits in base 10.

scale

The number of digits to the right of the decimal point.

sign

The sign is 1 for positive numbers, and 0 for negative numbers.

val

A number stored as a 16-byte scaled integer, with the least significant byte on the left.

For example, to specify the base 10 number 20.003 with a scale of 4, the number is scaled to an integer of 200030 (20.003 shifted by four tens digits), which is 186AA in hexadecimal. The value stored in the 16-byte integer is 5E 0D 03 00 00 00 00 00 00 00 00 00 00 00 00 00, the precision is the maximum precision, the scale is 4, and the sign is 1.

Character Code Conversions

Character code conversions under the OLE DB Provider for AS/400 and VSAM are controlled by a hierarchy of parameters. When connecting to mainframes all of these parameters are controlled on the personal computer. However, when connecting to data sources on the AS/400, parameter settings on the AS/400 as well as parameter settings on the personal computer can be involved.

The character code set identifier (CCSID) used by the host for a data source can be specified in several locations. The Host CCSID setting used by the OLE DB provider must be set to match the CCSID actually used on the host; otherwise data loss will occur. Note that some AS/400 systems default to a CCSID of 937 rather than 37 for enabling double-byte character sets (DBCS).

The following table illustrates the separate hierarchy controlling the Host CCSID parameter on the SNA client, AS/400 host, and mainframe host.

SNA client	**AS/400 host**	**Mainframe host**
SNA OLE DB provider (defaults to U.S./Canada 37)	System (DSPSYSVAL, QCCSID)	None. Determined by the Data Description (HCD file) on the SNA Client.
Data Source (the Host CCSID parameter in the Data Links file describing the Data Source)	User identifier (wrk usrprf)	
Data Description (HCD file)	Job (dspjob, crtjobd)	
	File (dspfd, crtpf)	
	Column (dspffd)	

Note

Only those fields in a Data Source that contain character data are affected by the Host CCSID parameters and character conversions. This section contains:

- Host CCSID and SNA OLE DB Provider
- Host CCSID and Data Source
- Host CCSID and Data Description
- Host CCSID and the Process Binary As Character Parameter

Host CCSID and SNA OLE DB Provider

The Host CCSID setting at the OLE DB provider level defaults to U.S./Canada (37).

Host CCSID and Data Source

The Host CCSID setting at the Data Source level is configured using Data Links for each Data Source. The **Host CCSID** parameter is configured under the **All** tab of the **Data Links** dialog box.

Valid values for the Host CCSID registry setting are any CCSID, including 65535. If the Host CCSID attribute is set to 65535, no character conversion occurs (the data is treated as binary). If the Host CCSID setting at the Data Source level does not exist, the value for Host CCSID defaults to the value at the SNA OLE DB provider level.

Host CCSID and Data Description

A Host CCSID attribute can be applied at the Data Description level. Each column in a host column description (HCD) file can have a Host CCSID attribute that determines how the character data in the column is to be converted. These attributes in the HCD file at the Data Description level should be configured using the SNA OLE DB management console snap-in. (For more information, see Configuring Data Descriptions) Valid values are any CCSID including 65535.

The Host CCSID attribute at the column Data Description level can be any value and may be empty. A Host CCSID value at the Data Description level overrides the value specified at the Data Source and OLE DB provider levels. If the Host CCSID is blank, the value for Host CCSID defaults to the value at the Data Source level. If the Host CCSID attribute is set to 65535, no character conversion occurs (the data is treated as binary).

Host CCSID and the Process Binary As Character Parameter

There is a Data Source parameter configurable using Data Links that affects whether binary data is considered as character data and is converted based on the Host CCSID setting. This **Process Binary As Character** parameter defaults to *false*. If this parameter is *false*, binary data is not treated as character (binary data is not affected by the Host CCSID setting). If this parameter is set to *true*, binary data is converted based on the Host CCSID setting.

This parameter is configured for each Data Source using Data Links under the **All** tab of the **Data Links** dialog box.

Using Package Designer with the OLE DB Provider for AS/400 and VASM

When using SQL Server Integration Services with the Microsoft OLE DB Provider for AS/400 and VSAM within the Package Designer in Visual Studio 2005, SSIS may raise the following warning dialog:

```
Warning at {9C6AC00C-13CF-4EF8-B44A-72055CC508C2} [OLE DB Source [262]]: Cannot retrieve the column code page info from the OLE DB provider. If the component supports the "DefaultCodePage" property, the code page from that property will be used. Change the value of the property if the current string code page values are incorrect. If the component does not support the property, the code page from the component's locale ID will be used.
```

Server Integration Services supports the option of specifying a per-column Locale Identifier for string data types, such as CHARACTER. However, the Microsoft OLE DB Provider for AS/400 and VSAM does not support this option. You can work around this limitation by modifying the settings in the Advanced Editor.

To allow the OLE DB Provider for AS/400 and VASM to use the default code page

1. In the Data Flow Task Design surface, add a new OLE DB Source or OLE DB Destination for use with the Microsoft OLE DB Provider for AS/400 and VSAM.
2. Right-click the OLE DB source or destination object, and then click **Show Advanced Editor...**.
3. On the Advanced Editor screen, click the Component Properties page.
4. Set **AlwaysUseDefaultCodePage** to True.
5. Click **OK**.

   Clicking OK saves the settings for use with the current OLE DB source or destination object within the SSIS package.

See Also

Other Resources

OLE DB Provider for AS/400 and VSAM Programmer's Guide

OLE DB Provider for DB2 Programmer's Guide

Microsoft® OLE DB Provider for DB2 enables users to access IBM DB2 from within an OLE-aware application. The object linking and embedding database (OLE DB) is a standard set of interfaces that provides heterogeneous access to disparate sources of information located anywhere—file systems, e-mail folders, and databases. The OLE DB Provider for DB2 combines the universal data access of OLE DB with the IBM Distributed Relational Database Architecture (DRDA).

Organizations have invested in secure, robust, enterprise-wide data storage and management systems. DRDA is a set of rules for distributing or extending relational data from one computer to another, such as from a personal computer server to an IBM DB2 database server running on a mainframe or an AS/400 computer. By combining the OLE DB and DRDA architectures, Microsoft enables organizations to preserve their investments in existing data management infrastructure, while extending universal data access to all enterprise-wide data sources.

For application programming interface (API) references and other technical information about OLE DB providers, see theOLE DB Providers Programmer's Reference section of the software development kit (SDK).

For more information on using the OLE DB Provider for DB2, seethe OLE DB Provider for DB2 section of the Operations guide.

In This Section

- Goals of the OLE DB Provider for DB2
- Distributed Relational Database Architecture
- Platforms Supported by the OLE DB Provider for DB2
- OLE DB Provider for DB2 Requirements
- Configuring the OLE DB Provider for DB2

Goals of the OLE DB Provider for DB2

Relational database management systems (RDBMS) are one of the major sources of mission-critical information in today's enterprise organizations. Relational database technology enables departments and individual users to save their information in centrally managed database stores that can be easily maintained by the organization's information systems group. Specific query tools designed for accessing relational database systems have added greater flexibility and ease of access to this information.

These same enterprises rely on vast networks of personal computers to enable their users to achieve business goals. End users invariably rely on network e-mail; Microsoft® Windows® productivity applications, such as Microsoft Office; and personal database programs, such as Microsoft Access, to accomplish their daily tasks. It is essential for these same users to incorporate data stored in relational database systems into their regular correspondence, analysis, and reports.

The challenge you have is how to provide access to this valuable data without the effort involved in developing traditional database applications. Much of the renewed interest in improved access to data sources is a result of the growth in the use of Internet and Web technology as mechanisms for delivering information. Fast and inexpensive methods of accessing data stored in RDBMS systems are necessary to deliver modern, three-tiered information systems during this era of cost cutting and budget tightening. Additional uses of this relational database access include specific queries and Web-based reporting.

IBM DB2 is a popular RDBMS for a significant number of enterprise customers. Customers need a cost-effective and manageable means to integrate DB2 with Microsoft SQL Server™, Microsoft Internet Information Services (IIS), and Microsoft Office applications. The goal of the OLE DB Provider for DB2 is to provide customers and solution providers with the means to integrate desktop applications with this wealth of data residing on DB2 database systems.

Distributed Relational Database Architecture

Database technology has enabled departments and individual users to save their information locally—as opposed to centrally managed stores owned by the organization's information systems group. Along with local storage and database query tools comes greater flexibility and ease of access to information. As more databases became distributed, a need emerged for users to access data stored remotely. IBM devised the Distributed Relational Database Architecture (DRDA) to enable their customers to access remote, distributed database systems across hardware platforms.

DRDA supports most dialects of the Structured Query Language (SQL) for access to relational database management systems (RDBMS). SQL is an international standard that defines a standardized language for accessing database management systems (DBMS). DRDA implementations generally support SQL in two ways: static (embedded) SQL, where the SQL commands are embedded directly in the application program and prepared as an extra step in the process of compiling the application; and dynamic or interactive (callable) SQL, where the user passes SQL commands as function calls at run time. One popular IBM implementation of dynamic SQL is the Call Level Interface (CLI). With dynamic SQL or CLI, SQL preparation is not required.

Clients that comply with DRDA are referred to as application requesters (AR) because they request data from the DRDA server. Servers that comply with DRDA are referred to as application servers (AS). Typically, application servers are implemented as the link to the RDBMS. In some cases, products are implemented as both application requesters and application servers.

DRDA supports access to stored procedures on DB2. SQL applications can invoke stored procedures or user-written programs on DB2 using the SQL **CALL** statement.

The OLE DB Provider for DB2 is an application requester implementation that can initiate DRDA commands to be serviced by a remote target DRDA application server represented by IBM DB2. On the Microsoft® Windows Server™ 2003 and Windows® 2000 Server operating systems, the Microsoft DRDA application requester can run as a service. This enables the integration of the DRDA service with other host applications using the IBM DRDA protocols and DRDA servers resident on the host. Microsoft host software is not required. (For more information, see Platforms Supported by the OLE DB Provider for DB2). IBM offers DB2 servers for most popular environments.

Platforms Supported by the OLE DB Provider for DB2

IBM and other software vendors have implemented Distributed Relational Database Architecture (DRDA) support into database systems, such as IBM DB2, and database tools on a wide range of operating systems. DRDA is an open, published, and widely supported protocol, which requires no additional license for development. This makes DRDA appealing to independent software vendors (ISVs), solution providers, large corporate development groups, as well as their customers.

The Microsoft® OLE DB Provider for DB2 is implemented as an IBM DRDA application requester, which means it connects to popular DRDA-compliant DB2 systems.

The Microsoft OLE DB Provider for DB2 can access DB2 systems through SNA LU 6.2 using Microsoft Host Integration Server, and can access DB2 systems directly using TCP/IP.

OLE DB Provider for DB2 Requirements

When connecting over SNA using LU 6.2, the OLE DB Provider for DB2 requires the following computer-to-host connectivity software:

- Microsoft Host Integration Server 2009
- Microsoft Host Integration Server Client

When connecting to a host system using TCP/IP, the OLE DB Provider for DB2 does not require any special host connectivity software.

The OLE DB Provider for DB2 supports the following OLE DB and ADO versions:

- OLE DB version 2.5. The Host Integration Server data access features require the run-time libraries for OLE DB version 2.5. These OLE DB libraries are installed as part of the Windows Server 2003 and Windows 2000 operating systems.
- ADO version 2.5. The Host Integration Server data access features require the run-time libraries for ADO version 2.5. On Windows Server 2003 and Windows 2000, these ADO libraries are installed as part of the Windows Server 2003 and Windows 2000 operating systems.

Configuring the OLE DB Provider for DB2

Microsoft® Data Access Components (MDAC) include Data Links, a generic method for managing and loading connections to OLE DB data sources. Microsoft Data Links provides a uniform method of creating persistent OLE DB data source object definitions stored in the form of universal data link (.udl) files. The OLE DB Provider for DB2 normally uses Data Links and .udl files for loading and configuring data sources.

The Data Source Wizard in the Microsoft Data Access Tool can help to define .udl files. OLE DB consumer applications, such as Data Transformation Services in Microsoft SQL Server™ can use the .udl files to connect to IBM data sources, such as DB2 and the mainframe file system.

To use Microsoft OLE DB Provider for DB2 with an OLE DB consumer application, the user must either create a Microsoft universal data link (.udl) file and call this from the application, or call the OLE DB provider from within the application using a connection string that includes the provider name and any other needed parameters.

This section contains:

Creating Data Links for the OLE DB Provider for DB2

Data source information must be configured for each DB2 system data source object that is to be accessed using the Microsoft® OLE DB Provider for DB2. The default parameters for the OLE DB Provider for DB2 are used as the default values for data sources when these parameters are not configured for each data source.

The Microsoft Data Links, a core element of the Microsoft Data Access Components, provides a uniform method for creating file-persistent OLE DB data source object definitions in the form of universal data link (.udl) files.

The Data Source Wizard in the Microsoft Data Access Tool can help to define .udl files. OLE DB consumer applications, such as Data Transformation Services in Microsoft SQL Server can use the .udl files to connect to IBM data sources, such as DB2 and the mainframe file system.

Creating New Data Links for the OLE DB Provider for DB2

You create new Data Links using the Data Source Wizard from the Data Access Tool.

Browsing Data Sources for the OLE DB Provider for DB2

You can browse data sources using the Data Source Browser window in the Data Access Tool.

To find the physical location of the .udl file associated with a data source, right-click the **DB2 OLE DB UDLs** folder, and then click **Locate** on the context menu. Windows Explorer opens which displays the location of the file.

Configuring Data Links for the OLE DB Provider for DB2

To edit the properties of a universal data link (.udl) file, right-click the file using Windows Explorer and click **Properties**. The **Properties** dialog box appears with property tabs:

- **General**
- **Security**
- **Summary**
- **Provider**
- **Connection**
- **Advanced**
- **All**

The **General**, **Security**, and **Summary** tabs provide access to general file information for the .udl file that is available for other files and is not related to the Data Link properties. This information includes file location, file type, file size, file dates, file security permissions for access, and descriptive summary information (description and origin properties and values such title, subject, and author) for the .udl file. The **General** tab has a text box with the name of the Data Link. This filename must end with the .udl extension if the file is to be recognized as a Data Link file. Note that the **Security** and **Summary** tabs are available on NTFS files systems, and not on the older FAT file systems.

The **Provider**, **Connection**, **Advanced**, and **All** tabs provide access to the Data Link properties that need to be configured to connect to the DB2 system.

The NewSnaDS tool can also be used to open and modify an existing .udl file. The **Data Link Properties** dialog box appears with property tabs.

This section contains:

Provider

The **Provider** tab enables you to select the OLE DB provider (the provider name string) to use in the universal data link (.udl) file from a list of possible OLE DB providers. Select **the Microsoft OLE DB Provider for DB2**. The parameters and fields displayed in the remaining tabs (**Connection**, **Advanced**, and **All**) are determined by the OLE DB provider that is selected.

Connection

The **Connection** tab enables you to configure the basic properties required to connect to a data source. The **Connection** tab dialog box contains several sections:

- Data source and Network connectivity
- Authentication
- Database Properties

For the Microsoft OLE DB Provider for DB2, the **Connection** tab includes the following properties for Data source and Network connectivity values.

Property	Description
Data source	The data source is an optional property that can be used to describe the data source.
Network	This drop-down list box allows selecting the type of network connection to be used. The allowable options are **TCP/IP Connection** or **APPC Connection**. If **TCP/IP Connection** is selected, click **More Options** () to open a dialog box for configuring TCP/IP network settings. The properties you can configure include the IP address of the DB2 host (or a hostname alias for this computer) and the Network Port (TCP/IP port) used for communication with the host. The default value for the Network Port is 446. The IP address of the host has no default value. If **APPC Connection** is selected (using SNA LU 6.2), click **More Options** () to open a dialog box for configuring APPC network settings. The properties you can configure include: the APPC local LU alias, the APPC remote LU alias, and the APPC mode name used for communication with the host. The default value for the APPC mode normally defaults to QPCSUPP. The local and remote LU alias fields do not have default values. The default value for the APPC mode name normally defaults to QPCSUPP. The APPC mode name can be selected from the drop-down list box.

The **Data source** in OLE DB is similar to a Data Source Name (DSN) in ODBC. The data source information is stored in a Microsoft Data Links file and contains the connection information required for the OLE DB Provider for DB2 to access IBM DB2.

For the Microsoft OLE DB Provider for DB2, the **Connection** tab includes the following properties for authentication information:

Property	Description
Single sign-on	Click this check box to enable using the Host Integration Server security features providing a Single Sign-On to access this OLE DB data source. When this check box is selected, the **User** name and **Password** fields are dimmed and inaccessible. The user name and password fields are set based on the logon name used for the Windows Server 2003 or Windows 2000 domain logon. When this check box is not selected, the **User** name and **Password** fields must normally contain appropriate values to access data sources on hosts.

User name	A valid user name and password are normally required to access data sources on a host. These values are case-sensitive. Users must not check the **Single sign-on** option if a specific user name and password are to be entered.
Password	A valid user name and password are normally required to access data sources on hosts. These values are case-sensitive. The **Blank password** check box is only applicable for a test connection. To enter a password, the user will need to clear the **Blank password** check box if it is selected. If **Blank password** is selected, a test connection with a blank password does not cause the OLE DB provider to prompt for a password. Optionally, users can choose to save the password in the .udl file by clicking the **Allow saving password** check box. Users and administrators should be warned that this option saves the authentication information (password) in plain text within the .udl file.

The AS/400 computer requires that the **User name** and **Password** properties be in uppercase. When connecting to DB2/400, these parameters must be passed as uppercase strings. When connecting to DB2 on IBM mainframes, the **User name** and **Password** parameters can be in mixed case.

It is possible to connect using a specific user name and password defined in DB2 on the host system or use the Single Sign-On (SSO) feature (often referred to as integrated Windows security). If a specific DB2 user name and password is to be used, this information may need to be saved into the .udl file. The user name and password are saved in plain text in the .udl file. For security reasons in these cases, it is imperative that the .udl file be protected with an access control list (ACL) that restricts access to only authorized users. Saving the user name and password in the data link also forces this .udl file to be updated whenever the password associated with the user name is changed. So for a variety of reasons, specifying a user name and password is not the preferred authentication option. Using the **Single Sign-On** option is the preferred method for authentication.

For the Microsoft OLE DB Provider for DB2, the **Connection** tab includes the following properties for database property values.

Property	Description
Initial catalog	This field is the first entry in the Database section of the **Connection** properties. This OLE DB property is used as the first part of a three-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, one can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE. If the provider supports changing the catalog for an initialized data source, the consumer can specify a different catalog name through the DBPROP_CURRENTCATALOG property in the DBPROPSET_DATASOURCE property set after initialization. This is a required property. This property is equivalent to the DBPROP_INIT_CATALOG OLE DB property ID.

Package collection	The name of the DRDA target collection (AS/400 library) where the Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. Package names are not restricted and can be uppercase, lowercase, or mixed case. The OLE DB provider creates packages dynamically in the location to which the user points using the Package Collection property. By default, the OLE DB provider will automatically create one package in the target collection, if one does not exist, at the time the user issues their first SQL statement. The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the User ID value configured in the data source. The package is created for use by SQL statements issued under the same isolation level specified when calling the OLE DB **ITransactionLocal::StartTransaction** or **ITransactionJoin::JoinTransaction** methods, as well as when setting the ADO **IsolationLevel** property on the **Connection** object. A problem can arise in multi-user environments. For example, if a user specifies a Package Collection value that represents a DB2 collection used by multiple users, but this user does not have authority to GRANT execute rights to the packages to other users (the PUBLIC group on the DB2 system, for example), the package is created for use only by this user. This means that other users may be unable to access the required package. The solution is for an administrative user with package administrative rights to create a set of packages for use by all users. (For more information, see Creating Packages for Use with the OLE DB Provider for DB2). The OLE DB Provider for DB2 ships with a tool program for use by administrators to create packages. The Data Access Tool is used to create packages. This tool can be run using a privileged User ID to create packages in collections accessed by multiple users. This tool creates a set of packages and grants **EXECUTE** privilege on these packages to the PUBLIC group representing all users on the DB2 system. The packages (for more information, see descriptions under the isoLevel parameter of the OLE DB **ITransactionLocal::StartTransaction** or **ITransactionJoin::JoinTransaction** methods, as well as the ADO IsolationLevel property) created are as follows: AUTOCOMMITTED package (MSNC001 is only applicable on DB2/400) READ UNCOMMITTED package (MSUR001) READ COMMITTED package, (MSCS001) REPEATABLE READ package, (MSRS001) SERIALIZABLE package (MSRR001) Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400. Once created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, the DB2 for OS/400 QSYS2.SYSPACKAGE, and the DB2 Universal Database (UDB) SYSIBM.SYSPACKAGE catalog tables. Note that when upgrading from SNA Server 4.0, any existing SNA 4.0 packages must be re-created using the Host Integration Server CrtPkg tool to make them compatible with Host Integration Server 2009. The package names changed from SNA Server 4.0. This property is equivalent to the DBPROP_DB2OLEDB_PACKAGECOL OLE DB property ID.
Default schema	The name of the collection where the OLE DB Provider for DB2 looks for catalog information. The Default Schema is the SCHEMA name for the target collection of tables and views. The OLE DB Provider uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection. For DB2, the Default Schema is the target AUTHENTICATION (User ID or owner). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, the OLE DB provider uses the USER_ID provided at logon. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Because this default is inappropriate in many cases, it is essential that the Default Schema value in the data source be defined. This property is equivalent to the DBPROP_DB2OLEDB_CATALOGCOL OLE DB property ID.

The **Connection** tab also includes a **Test Connection** button that can be used to test the connection properties. The connection can only be tested after all of the required parameters are entered. When this button is pressed, an APPC session or a TCP/IP session attempts to be established with the host using the OLE DB Provider for DB2.

Advanced

The **Advanced** tab enables users to select the character code set identifier used by the host, the personal C code page used on the client, and some specific options when using the OLE DB Provider for DB2.

For the Microsoft OLE DB Provider for DB2, these properties include the following values.

Property	Description
DBMS Platform	The target DB2 platform property value is used to optimize performance of the OLE DB provider when executing operations such as data conversion. The default value is DB2/MVS.
Default Qualifier	The name of the schema (collection/owner) with which to fully qualify unqualified object names. This attribute enables the user to access database objects without fully qualifying the objects using a collection (schema) qualifier. The OLE DB provider sends this value to DB2 using a **SET CURRENT SQLID** statement, instructing the DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statements. If you do not set a value for default qualifier, no **SET** statement is issued by the OLE DB Provider. This OLE DB property is only valid when connecting to DB2 for MVS (OS/390, z/OS).
Host CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the Process Binary as Character value is set to **true**, character data is converted based on the DB2 column CCSID and default ANSI code page. This property defaults to U.S./Canada (37). This property is equivalent to the DBPROP_DB2OLEDB_HOSTCCSID OLE DB property ID.
PC code page	The PC code page property indicates the code page to be used on the computer for character code conversion. This property is required when processing binary data as character data. Unless the **Process binary as character** check box is selected (value is set to **true**), character data is converted based on the default ANSI code page configured in Windows. This property defaults to Latin 1 (1252). This property is equivalent to the DBPROP_DB2OLEDB_PCCODEPAGE OLE DB property ID.
Read only	When this option is selected, the OLE DB Provider for DB2 creates a read-only data source by setting the Mode property to Read (DB_MODE_READ). A user has read access to objects such as tables, and cannot do update operations (**INSERT**, **UPDATE**, or **DELETE**, for example). This property defaults to a **Mode** property of Read/Write (DB_MODE_READ/WRITE). This property is equivalent to the DBPROP_INIT_MODE OLE DB property ID.
Process binary as character	When this option is selected (property is set to **true**), the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This property defaults to **false**. This property is equivalent to the DBPROP_DB2OLEDB_BINASCHAR OLE DB property ID.
Distributed transactions	When this option is selected, two-phase commit (distributed unit of work) is enabled. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and a Host Integration Server 2009 Resync Service. This option requires that an **APPC Connection** is selected as the network transport in the **Connection** tab and Microsoft Transaction Server (MTS) is installed. Note that Host Integration Server supports 2PC over TCP/IP as well as SNA LU 6.2. This property is equivalent to the DBPROP_DB2OLEDB_UNITSOFWORK OLE DB property ID.

All

The **All** tab enables users to configure essentially all of the properties for the data source except for the OLE DB provider. The properties available in the **All** tab include properties that can be configured using the **Connection** and **Advanced** tabs as well as optional detailed properties used to connect to a data source.

For the Microsoft OLE DB Provider for DB2, these properties include the following values.

Property	Description
Affiliate Application	This property is only used when Single Sign-On is enabled. It contains the application name to use when retrieving host credentials from the Single Sign-On database. This property is equivalent to the DBPROP_DB2OLEDB_AFFILIATEAPP OLE DB property ID.
Alternate TP Name	The Alternate Transaction Program (TP) Name property represents the default transaction program name for the DB2 DRDA application server (AS), which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server 2009 uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, the Alternative TP Name is set to 0X07F9F9F9. This property is equivalent to the DBPROP_DB2OLEDB_TPNAME OLE DB property ID.
APPC Local LU Alias	When an **APPC Connection** using SNA LU 6.2 is selected for the Network Transport Library, this field is the name of the local LU alias configured in SNA services. This property is equivalent to the DBPROP_DB2OLEDB_LOCALLU OLE DB property ID.
APPC Mode Name	When an **APPC Connection** using SNA LU 6.2 is selected for the Network Transport Library, this field is the APPC mode and must be set to a value that matches the host configuration and SNA services configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This property normally defaults to QPCSUPP. This property is equivalent to the DBPROP_DB2OLEDB_APPCMODE OLE DB property ID.
APPC Remote LU Alias	When an **APPC Connection** using SNA LU 6.2 is selected for the Network Transport Library, this field is the name of the remote LU alias configured in SNA services. This property is equivalent to the DBPROP_DB2OLEDB_REMOTELU OLE DB property ID.

APPC Security Type	When an **APPC Connection** using SNA LU 6.2 is selected for the Network Transport Library, this field is the security type that is used when allocating an SNA connection. The allowable values are: **Program** (default) and **Same**. This property is equivalent to the DBPROP_DB2OLEDB_APPCSEC OLE DB property ID.
Cache Authentication	This property determines whether the OLE DB provider caches authentication information. This property defaults to **false**. The value of this property (**true** or **false**) is selected from the drop-down list box. This property is equivalent to the DBPROP_AUTH_CACHE_AUTHINFO OLE DB property ID.
Client Application Name	Use this property to specify a custom application name that the DB2 Network Client (DRDA AR) will pass to the DB2 Server when connecting. When troubleshooting issues, you can use this value to help identify relevant portions within DB2 or system logs. This property is equivalent to the DBPROP_DB2OLEDB_CLIENTAPPNAME OLE DB property ID.
Connection Pooling	This property determines whether connection pooling is used. Connection pooling keeps a connection active after it is closed in case another connection is opened with the same properties. The default value is **False**. This property is equivalent to the DBPROP_DB2OLEDB_CONNPOOLING OLE DB property ID.
Data Source	The data source is an optional property that can be used to describe the data source. This property does not have a default value.
DBMS Platform	The target DB2 platform property value is used to optimize performance of the OLE DB provider when executing operations such as data conversion. The default value is DB2/MVS.
Default Qualifier	The name of the schema (collection/owner) with which to fully qualify unqualified object names. This attribute enables the user to access database objects without fully qualifying the objects using a collection (schema) qualifier. The OLE DB provider sends this value to DB2 using a **SET CURRENT SQLID** statement, instructing the DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statements. If you do not set a value for default qualifier, no **SET** statement is issued by the OLE DB Provider. This OLE DB property is only valid when connecting to DB2 for MVS (OS/390, z/OS).
Default Schema	The name of the collection where the OLE DB Provider for DB2 looks for catalog information. The Default Schema is the SCHEMA name for the target collection of tables and views. The OLE DB Provider uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection. For DB2, the Default Schema is the target AUTHENTICATION (User ID or owner). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, the OLE DB Provider uses the USER_ID provided at logon. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Because this default is inappropriate in many cases, it is essential that the Default Schema value in the data source be defined. This property is equivalent to the DBPROP_DB2OLEDB_CATALOGCOL OLE DB property ID.

Extended Properties	This property is a string containing provider-specific, extended connection information. The use of this property implies that the OLE DB consumer knows how this string will be interpreted and used by the OLE DB provider. This parameter should be used only for provider-specific connection information that cannot be explicitly described through the other property values. This property is equivalent to the DBPROP_INIT_PROVIDERSTRING OLE DB property ID.
Host CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote host computer. The CCSID property is required when processing binary data as character data. Unless the Process Binary as Character value is set to true, character data is converted based on the DB2 column CCSID and default ANSI code page. This property defaults to U.S./Canada (37). This property is equivalent to the DBPROP_DB2OLEDB_HOSTCCSID OLE DB property ID.
Initial Catalog	This OLE DB property is used as the first part of a three-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, one can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE. If the provider supports changing the catalog for an initialized data source, the consumer can specify a different catalog name through the DBPROP_CURRENTCATALOG property in the DBPROPSET_DATASOURCE property set after initialization. This is a required property. This property is equivalent to the DBPROP_INIT_CATALOG OLE DB property ID.
Integrated Security	This property determines whether the OLE DB provider uses Host Security Integration (Single Sign-On). When this property is set to SSPI, Single Sign-On is enabled and separate user ID and password parameters are not required. The user name and password fields are set based on the logon name used for the Windows Server 2003 or Windows 2000 domain logon. When this property is null, this Single Sign-On feature is disabled. This property defaults to null (host security integration is disabled) and a user ID and password are required. This property is equivalent to the DBPROP_AUTH_INTEGRATED OLE DB property ID.

Mode	A Mode property is a bitmask specifying access permissions. This bitmask can be a combination of zero or more of the following: DB_MODE_READ—Read-only. DB_MODE_READWRITE—Read/write (DB_MODE_READ | DB_MODE_WRITE). DB_MODE_SHARE_DENY_NONE—Neither read nor write access can be denied to others. DB_MODE_SHARE_DENY_READ—Prevents others from opening in read mode. DB_MODE_SHARE_DENY_WRITE—Prevents others from opening in write mode. DB_MODE_SHARE_EXCLUSIVE—Prevents others from opening in read/write mode (DB_MODE_SHARE_DENY_READ | DB_MODE_SHARE_DENY_WRITE). DB_MODE_WRITE—Write-only. The following values for mode are supported by the OLE DB Provider for DB2: Read (DB_MODE_READ) and Read/Write (DB_MODE_READ/WRITE). This property defaults to Read/Write. When the **Read Only** property is selected in the **Advanced** tab, the OLE DB Provider for DB2 creates a read-only data source by setting the **Mode** property to Read (DB_MODE_READ). A user has read access to objects such as tables, and cannot do update operations (**INSERT**, **UPDATE**, or **DELETE**, for example). This property is equivalent to the DBPROP_INIT_MODE OLE DB property ID.
Network Address	When TCP/IP has been selected for the Network Transport Library, this property indicates the IP address of the DB2 host or a hostname alias for this computer. This property is equivalent to the DBPROP_DB2OLEDB_NETADDRESS OLE DB property ID.
Network Port	When TCP/IP has been selected for the Network Transport Library, this property is the TCP/IP port used for communication with the DB2 host. The default value is TCP/IP port 446. This property is equivalent to the DBPROP_DB2OLEDB_NETPORT OLE DB property ID.
Network Transport Library	The network transport dynamic-link library property designates whether the OLE DB Provider for DB2 connects through an APPC connection using SNA LU 6.2 or a TCP/IP connection. The possible values for this property are TCPIP or SNA. The default value for this property is **SNA**. If the default SNA is selected, values for APPC Local LU Alias, APPC Mode Name, and APPC Remote LU Alias are required. If TCPIP is selected, values for Network Address and Network Port are required. This property is equivalent to the DBPROP_DB2OLEDB_NETTYPE OLE DB property ID.
New Password	This property allows an OLE DB connection to change the host password for the user account used for the connection. If this property is set, the account password will be changed to the property value. This property is equivalent to the DBPROP_DB2OLEDB_NEWPWD OLE DB property ID.

Package Collection	The name of the DRDA target collection (AS/400 library) where the Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. Package names are not restricted and can be uppercase, lowercase, or mixed case. The OLE DB provider creates packages dynamically in the location to which the user points using the Package Collection property. By default, the OLE DB provider automatically creates one package in the target collection, if one does not exist, at the time the user issues the first SQL statement. The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the User ID value configured in the data source. The package is created for use by SQL statements issued under the same isolation level specified when calling the OLE DB **ITransactionLocal::StartTransaction** or **ITransactionJoin::JoinTransaction** methods, as well as when setting the ADO **IsolationLevel** property on the **Connection** object. A problem can arise in multi-user environments. For example, if a user specifies a Package Collection value that represents a DB2 collection used by multiple users, but this user does not have authority to GRANT execute rights to the packages to other users (the PUBLIC group on the DB2 system, for example), the package is created for use only by this user. This means that other users may be unable to access the required package. The solution is for an administrative user with package administrative rights to create a set of packages for use by all users. (For more information, see **Creating Packages for Use with the OLE DB Provider for DB2**.) The OLE DB Provider for DB2 ships with a tool program for administrators to create packages. You use the **Data Access Tool** to create packages. This tool can be run using a privileged User ID to create packages in collections accessed by multiple users. This tool creates a set of packages and grants EXECUTE privilege on these packages to the PUBLIC group representing all users on the DB2 system . The packages (for more information, see descriptions under the isoLevel parameter of the OLE DB **ITransactionLocal::StartTransaction** or **ITransactionJoin::JoinTransaction** methods, as well as the ADO **IsolationLevel** property) created are as follows: AUTOCOMMITTED package (MSNC001 is only applicable on DB2/400) READ UNCOMMITTED package (MSUR001) READ COMMITTED package, (MSCS001) REPEATABLE READ package, (MSRS001) SERIALIZABLE package (MSRR001) Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400. Once created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, the DB2 for OS/400 QSYS2.SYSPACKAGE, and the DB2 Universal Database (UDB) SYSIBM.SYSPACKAGE catalog tables. Note that when upgrading from SNA Server 4.0, any existing SNA 4.0 packages must be re-created using the Host Integration Server **Data Access Tool** to make them compatible with Host Integration Server 2009. The package names changed from SNA Server 4.0. This property is equivalent to the DBPROP_DB2OLEDB_PACKAGECOL OLE DB property ID.
Password	A valid user name and password are normally required to access data sources on hosts. The password is case-sensitive and is displayed as asterisks in this dialog box for security purposes. This property is equivalent to the DBPROP_AUTH_PASSWORD OLE DB property ID.
PC Code Page	The PC Code Page property indicates the code page to be used on the personal computer for character code conversion. This property is required when processing binary data as character data. Unless the Process Binary as Character value is set to **true**, character data is converted based on the default ANSI code page configured in Windows. This property defaults to Latin 1 (1252). This property is equivalent to the DBPROP_DB2OLEDB_PCCODEPAGE OLE DB property ID.

Persist Security Info	This property indicates whether the data source object is allowed to persist sensitive authentication information, such as a password along with other authentication information. This property defaults to **false**. The value of this property (**true** or **false**) is selected from the drop-down list box. This property is equivalent to the DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO OLE DB property ID.
Process Binary as Character	When this property is set to true, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This property defaults to **false**. The value of this property (**true** or **false**) is selected from the drop-down list box. This property is equivalent to the DBPROP_DB2OLEDB_BINASCHAR OLE DB property ID.
Units of Work	This property indicates whether two-phase commit (distributed unit of work) used for transactions is supported for this data source. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the Host Integration Server Resync Service. The following values for this property are supported by the OLE DB Provider for DB2: RUW (remote unit of work) DUW (distributed unit of work) This property defaults to RUW. Distributed unit of work (two-phase commit) requires an APPC Connection using SNA LU 6.2 or 2PC over TCP/IP as the network transport. Microsoft Transaction Server (MTS) should also be installed. This property is equivalent to the DBPROP_DB2OLEDB_UNITSOFWORK OLE DB property ID.
User ID	A valid user name is normally required to access data sources on hosts. This value is case-sensitive. This property is equivalent to the DBPROP_AUTH_USERID OLE DB property ID.

These properties on the **All** tab may be edited by selecting a property from the list displayed and selecting **Edit Value**. This button invokes a dialog box for the specific property containing a property description describing the property and a property value box for making changes.

Creating Packages for Use with the OLE DB Provider for DB2

The Microsoft OLE DB Provider for DB2, which is implemented as an IBM Distributed Relational Database Architecture (DRDA) application requester, uses packages to issue SQL statements and call DB2 stored procedures. There is a provider-specific property that the OLE DB Provider for DB2 uses to identify a location in which to create and store DB2 packages. The OLE DB Provider for DB2 creates packages dynamically in the location to which the user points using the Package Collection property corresponding to the DBPROP_DB2OLEDB_PACKAGECOL property ID of OLE DB. This location may be configured using the **Connection** and **Advanced** tabs using Microsoft Data Links or can be passed as part of the connection string as an attribute keyword and argument. This attribute keyword can be either pkgcol or the long form of this attribute, Package Collection.

There are two package creation options:

1. The OLE DB Provider for DB2 autocreates one package for the currently used isolation level at run time if no package already exists. This autocreate process may fail if the user account does not have authority to create packages.

2. An administrator or user can manually creates all four packages (five packages on DB2/400) for use with all isolation levels and for use by all users (the PUBLIC group on DB2 representing all users) or a specific set of users. The OLE DB Provider for DB2 includes a program for use by users with appropriate administrative privilege that will create these packages and grant access to the PUBLIC group for this purpose.

However, some users may not have the security level when manually creating packages to GRANT authority to the packages to other users (grant authority to the DB2 PUBLIC group representing all users, for example). This can be a problem if two or more users with different user IDs try to access a single collection of packages. The first user that created the packages will have access to the packages, but the second user likely will not. The Host Integration Server 2009 CD includes a program for use by an administrator to create packages. This tool can be run using a privileged User ID to create packages in collections accessed by multiple users. You use the Data Access Tool to create packages for use with DB2.

A shortcut for this tool is added to the **Programs** menu off the **Start** button on the Windows taskbar under the **Host Integration Server\Data Integration** folder with a name of **Data Access Tool**. This shortcut is created when the Microsoft Host Integration Server or the Host Integration Client is first installed and support for Data Access is checked.

This tool creates a set of packages and grants EXECUTE privileges on these packages to the PUBLIC group. The PUBLIC group on DB2 systems is a default group that represents all DB2 users. The following packages are created:

- AUTOCOMMITTED package (MSNC001) is only applicable on DB2/400)
- READ UNCOMMITTED package (MSUR001)
- READ COMMITTED package (MSCS001)
- REPEATABLE READ package (MSRS001)
- SERIALIZABLE package (MSRR001)

Note that the AUTOCOMMITTED package (MSNNC001) is only created on DB2 for OS/400.

The descriptive process name used by the Data Access Tool corresponds with the isolation levels defined in the ANSI SQL standard. The following table indicates how these packages correspond with the terms used by IBM for isolation levels in DB2 documentation.

Package description	**Package name**	**IBM documentation**
AUTOCOMMITTED (Note that this applies only to DB2/400 and does not correspond with an ANSI SQL isolation level)	MSNC001	COMMIT(*NONE) (NC). This isolation level is used in DB2/400 autocommit mode only and has no corresponding isolation level on other DB2 platforms or in ANSI SQL.

READ UNCOMMITTED	MSUR001	UNCOMMITTED READ (UR). This isolation level corresponds with ANSI SQL READ UNCOMMITTED.
READ COMMITTED	MSCS001	CURSOR STABILITY (CS). This isolation level corresponds with ANSI SQL READ COMMITTED.
REPEATABLE READ	MSRS001	READ STABILITY (RS). This isolation level corresponds with ANSI SQL REPEATABLE READ.
SERIALIZABLE	MSRR001	REPEATABLE READ (RR). This isolation level corresponds with ANSI SQL SERIALIZABLE.

Note that when upgrading from SNA Server 4.0, any existing SNA 4.0 packages must be re-created using the Host Integration Server Data Access Tool to make them compatible with Host Integration Server 2009. The package names used by the OLE DB Provider for DB2 on SNA Server 4.0 are not compatible with the OLE DB Provider for DB2 included with Host Integration Server. On SNA Server 4.0, these packages used different names as follows:

```
AUTOCOMMITTED package (SNANC001) only applicable on DB2/400
READ UNCOMMITTED package (SNACH001)
READ COMMITTED package, (SNACS001)
REPEATABLE READ package, (SNARR001)
SERIALIZABLE package (SNAAL001)
```

These isolation levels are described in detail in Support for Isolation Levels Using the OLE DB Provider for DB2. These isolation levels are also described under the OLE DB isoLevel parameter and ADO IsolationLevel property. Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400.

Note that the Data Access Tool creates this set of packages and grants EXECUTE privileges to the PUBLIC group. There may be cases for security reasons where EXECUTE privileges to this set of packages on the DB2 system should be restricted to a different group of users or specific users. In these cases, execution privileges on these created packages need to be modified on the host system.

The Data Access Tool creates all of these packages inside the Collection that is specified in the Package Collection property in the data link file, or in the connection string. If the user does not have the appropriate authority to create packages in the specified Collection, or if the specified Collection does not exist, the OLE DB Provider for DB2 will return an error.

In the case of DB2 on MVS or OS/390, the normal error text returned if the user does not have the appropriate authority would be as follows:

```
A SQL error has occurred. Please consult the documentation for your specific DB2 version fo
r a description of the associated Native Error and SQL State. SQLSTATE: 51002, SQLCODE: -56
7.
```

In the case of DB2/400, the normal error text returned if the user does not have the appropriate authority would be as follows:

```
A SQL error has occurred. Please consult the documentation for your specific DB2 version fo
r a description of the associated Native Error and SQL State. SQLSTATE: 51002, SQLCODE: -80
5.
```

In the case of DB2/400, the normal error returned if the collection does not exist would be as follows:

```
Failed to create AUTOCOMMITTED (NC) package.  RETCODE=-99.
SQL Error: Code=-204, State=42704, Error Text= A SQL error has occurred.  Please consult th
e documentation for your specific DB2 version for a description of the associated Native Er
```

```
ror and SQL State. SQLSTATE: 42704, SQLCODE: -204
```

There are two authorities required to execute the create package process on MVS using the Data Access Tool:

```
GRANT BINDADD TO <authorization ID>
GRANT CREATE IN COLLECTION <collection ID> TO <authorization ID>
```

The "authorization ID" is the user who needs the permission to create the packages. The "collection ID" is the name of the collection, which the user specifies in the data link file for the Package Collection property. This collection should be a valid collection within the DB2. If an administrator executes the preceding statements on behalf of a nonprivileged user, this nonprivileged user can then run the CrtPkg tool. Once run, the CrtPkg process creates four sets of packages (one for each of the four isolation levels supported on DB2 for MVS or OS/390) for use by all (PUBLIC) users of the Microsoft data access features.

The following example illustrates this process on DB2 for MVS or DB2 for OS/390.

Grant rights to run the CrtPkg tool to authorization ID WNW999

```
GRANT BINDADD TO WNW999
GRANT CREATE IN COLLECTION MSPKG TO WNW999
```

Run the Data Access Tool using authorization ID WNW999.

To execute the Data Access Tool on DB2/400, a user ID must have one of the following authorities:

- *CHANGE authority on the DB2 collection
- *ALL authority on the DB2 collection

If the user has *USE authority or if the user has *EXCLUDE authority, the Create Package process will fail.

There are several steps required to change user authority on a DB2/400 collection (AS/400 library): From interactive SQL (STRSQL command) while logged on as user with administrative privileges, create a new collection. This command can also be issued using ADO, OLE DB, and ODBC. However, most administrators typically create collections from the AS/400 console because the administrator must be logged on at the console to issue the Command Language (CL) command with which to change the user authority on the collection.

```
CREATE COLLECTION <collection ID>
```

From the AS/400 command console, type the CL WRKOBJ command with the <collection ID> as a parameter.

```
WRKOBJ <collection ID>
```

The "collection ID" is the name of the collection, which the user specifies in the data link file for the Package Collection property. This collection should be a valid collection within DB2. The **Work with objects** dialog box appears. Place the cursor on the *PUBLIC Object Authority line and change the authority from *USE to *ALL.

If an administrator executes the preceding statements on behalf of a nonprivileged user, this nonprivileged user can then run the CrtPkg tool. Once run, the CrtPkg process creates five sets of packages (one for each of the five isolation levels supported on DB2/400) for use by all (PUBLIC) users of the Microsoft data access features. On DB2/400, five packages are created including the AUTOCOMMITTED packages.

The following example illustrates this process on DB2/400.

Grant rights to run the Data Access Tool to authorization ID WNW999

```
CREATE COLLECTION MSPKG
```

```
WRKOBJ MSPKG
```

Run the Data Access Tool.

When using the Data Access Tool, if the package collection specified does not exist, DB2 returns SQLCODE -805.

When using auto-create packages, if a package collection is not specified or the package collection does not exist, during the auto-create package process, the consumer application receives SQLSTATE HY000 and SQLCODE -385. The SQLSTATE HY000 is defined as a provider-specific error. The -385 Error Return Code is not a SQLCODE but rather a DDM DRDA AR (DB2 client) return code. This error code is defined as DDM_VALNSPRM with the following associated text string:

```
"The parameter value is not supported by the target system."
```

The OLE DB Provider for DB2 client error codes are defined in the Db2oledb.h file located on the Host Integration Server CD.

Note that when upgrading from SNA Server 4.0, any existing SNA 4.0 packages must be re-created using the Host Integration Server Data Access Tool to make them compatible with Host Integration Server.

ADO Object, Method, Property, and Collection Support for AS/400, VSAM and DB2

Microsoft® ActiveX® Data Objects (ADO) version 2.0 defines a number of objects, methods, properties, and collections.

Microsoft OLE DB Provider for AS/400 and VSAM supports the ADO objects, methods, properties, and collections that are appropriate for an OLE DB data provider accessing a non-SQL host file system.

Microsoft OLE DB Provider for DB2 supports the ADO objects, methods, properties, and collections that are appropriate for an OLE DB data provider accessing an SQL database.

Similar support for ODBC can be found in the topic ADO Object Support in the ODBC Driver for DB2.

This section contains:

- ADO Object Support in the OLE DB Provider for AS/400 and VSAM
- ADO Object Support in the OLE DB Provider for DB2

ADO Object Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the Microsoft® ActiveX® Data Objects (ADO) version 2.0 objects that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

ADO object	Support
Collection	Yes, most methods
Command Object	Yes, some methods, some properties, and all collections
Connection Object	Yes, some methods, some properties, and all collections
Error Object	Yes, some properties
Field Object	Yes, all methods, properties, and collections
Parameter Object	No
Recordset Object	Yes, most methods, most properties, and all collections

This section contains:

- ADO Method Support in the OLE DB Provider for AS/400 and VSAM
- ADO Property Support in the OLE DB Provider for AS/400 and VSAM
- ADO Collection Support in the OLE DB Provider for AS/400 and VSAM
- Command Object in the OLE DB Provider for AS/400 and VSAM (ADO)
- Connection Object in the OLE DB Provider for AS/400 and VSAM (ADO)
- Error Object in the OLE DB Provider for AS/400 and VSAM (ADO)
- Field Object in the OLE DB Provider for AS/400 and VSAM (ADO)
- Recordset Object in the OLE DB Provider for AS/400 and VSAM (ADO)

ADO Method Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the ActiveX® Data Objects (ADO) version 2.0 object methods that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

ADO object	Method	Support
Collection object	**Append** Method	No
	Clear Method	Yes
	Delete Method	Yes
	Item Method	Yes
	Refresh Method	Yes
Command Object	**CreateParameter** Method	No
	Cancel Method	No
	Execute Method	Yes, but options must be **adCmdText**
Connection Object	**BeginTrans** Method	No
	Cancel Method	No
	Close Method	Yes
	CommitTrans Method	No
	Execute Method	Yes, but options must be **adCmdText**
	Open Method	Yes
	OpenSchema Method	Yes
	RollbackTrans Method	No
Field Object	AppendChunk Method	Yes
	GetChunkMethod	Yes
	ReadFromFile Method	No
	WriteToFile Method	No
Parameter Method	**AppendChunk** Method	No
Recordset Object	AddNew Method	Yes
	Cancel Method	No
	CancelBatch Method	Yes
	CancelUpdate Method	Yes
	Clone Method	Yes
	Close Method	Yes
	Delete Method	Yes

	Find Method	Yes
	GetRows Method	Yes
	Move Method	Yes
	MoveFirst Method	Yes
	MoveLast Method	Yes
	MoveNext Method	Yes
	MovePrevious Method	Yes
	NextRecordset Method	No
	Open Method	Yes
	Requery Method	Yes
	Resync Method	No
	Save Method	Yes
	Seek Method	No
	Supports Method	Yes
	Update Method	Yes
	UpdateBatch Method	Yes

Note

The **Collection** object is a special case, representing a collection of other ADO objects. These collection objects support several methods:

- **Append** to add an object to a collection
- **Clear** to empty all objects from a collection
- **Delete** to remove a single object from a collection
- **Item** to return a specific member object of a collection by name or ordinal number
- **Refresh** to update the objects in a collection to reflect objects available from and specific to the OLE DB provider

ADO Property Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the ActiveX® Data Objects (ADO) version 2.0 object properties that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

ADO object	Property	Support
Command Object	ActiveConnection Property	Yes
	CommandText Property	Yes
	CommandTimeout Property	No
	CommandType Property	Yes
	Prepared Property	No
	State Property	Yes
Connection Object	Attributes Property	Yes
	CommandTimeout Property	No
	ConnectionString Property	Yes
	ConnectionTimeout Property	No
	CursorLocation Property	Yes
	DefaultDatabase Property	No
	IsolationLevel Property	No
	Mode Property	Yes
	Provider Property	Yes
	State Property	Yes
	Version Property	Yes
Error Object	Description Property	Yes
	HelpContext Property	No
	HelpFile Property	No
	NativeError Property	Yes
	Number Property	Yes
	Source Property	Yes
	SQLState Property	No
Field Object	ActualSize Property	Yes
	Attributes Property	Yes
	DataFormat Property	No
	DefinedSize Property	Yes

	Name Property	Yes
	NumericScale Property	Yes
	OriginalValue Property	Yes
	Precision Property	Yes
	Type Property	Yes
	UnderlyingValue Property	Yes
	Value Property	Yes
Parameter Object	**Attributes** Property	No
	Direction Property	No
	Name Property	No
	NumericScale Property	No
	Precision Property	No
	Size Property	No
	Type Property	No
	Value Property	No
Recordset Object	**AbsolutePage** Property	No
	AbsolutePosition Property	No
	ActiveCommand Property	Yes
	ActiveConnection Property	Yes
	BOF Property	Yes
	Bookmark Property	Yes
	CacheSize Property	Yes
	CursorLocation Property	Yes
	CursorType Property	Yes
	DataMember Property	No
	DataSource Property	No
	EditMode Property	Yes
	EOF Property	Yes
	Filter Property	Yes
	Index Property	No
	LockType Property	Yes
	MarshalOptions Property	No
	MaxRecords Property	No

	PageCount Property	No
	PageSize Property	No
	RecordCount Property	No
	Sort Property	Yes
	Source Property	Yes
	State Property	Yes
	Status Property	Yes
	StayInSync Property	No

ADO Collection Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the ActiveX® Data Objects (ADO) version 2.0 object collections that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

ADO object	Collection	Support
Command Object	**Parameters**	No
	Properties	Yes
Connection Object	**Errors**	Yes
	Properties	Yes
Field Object	**Properties**	Yes
Parameter	**Properties**	No
Recordset Object	**Fields**	Yes
	Properties	Yes

Command Object in the OLE DB Provider for AS/400 and VSAM (ADO)

The ActiveX® Data Objects (ADO) **Command** object is a definition of a specific command that is executed against an OLE DB data source.

You use **Command** objects to create a **Recordset** object and obtain records, execute a bulk operation, or manipulate the structure of a database. When using Microsoft OLE DB Provider for AS/400 and VSAM, some collections, methods, or properties of a **Command** object may generate an error when called.

The primary purpose of the **Command** object in the context of OLE DB Provider for AS/400 and VSAM is to issue AS/400 command language (CL) commands for execution by the remote OS/400 DDM target server. For a listing of legal distributed data management (DDM) command-line strings, see *AS/400 DDM User's Guide* published by IBM.

The following table lists the **Command** object methods, properties, and collections that the current version of OLE DB Provider for AS/400 and VSAM supports.

Name	Comment
Execute Method	Evaluates command text as a text string. (The only supported *Options* parameter for this method is **adCmdText**, which indicates that this is not an SQL command.)
ActiveConnection Property	Sets or returns the information used to establish a connection to a data source. (For more information, see the notes that follow.)
CommandText Property	Sets or returns the command text to be executed.
CommandType Property	Sets or returns the type of command in a **CommandText** property.
State Property	Describes the current state of an object.
Properties collection	Collections of properties on the command.

The **Execute** method executes a command and returns a **Recordset** object, if appropriate. You can use the **Command** object to open tables or execute DDM commands on a remote DDM server. If errors occur, they can be examined with the **Errors** collection on the **Connection** object.

A **Command** object can be created independently of a previously defined **Connection** object by setting the **ActiveConnection** property of the Command object to a valid connection string. ADO still creates a **Connection** object, but it does not assign that object to an object variable. However, if multiple **Command** objects are to be associated with the same connection, the **Connection** object needs to be explicitly created and opened. This assigns the **Connection** object to an object variable. If the **ActiveConnection** property of the **Command** object is not set to this object variable, ADO creates a new **Connection** object for each **Command** object, even if the same connection string is used.

The **ActiveConnection** property associates an open connection with a **Command** object. The **CommandText** property defines the text version of a command. The syntax for the string in the **CommandText** property when used with OLE DB Provider for AS/400 and VSAM is as follows:

```
EXEC COMMAND DDMCmd
```

Where **DDMCmd** represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands enable you to request functions from the OS/400 operating system. Some examples are the **DLTF** (Delete File) or **DSPFFD** (Display File Description) commands. These are the same commands that you could type at the command prompt if you were connected to an AS/400 computer through a 5250 terminal session. For a detailed list of possible commands, see the OS/400 CL Reference for your platform.

The **CommandType** property specifies the type of command described in the **CommandText** property prior to execution to optimize performance. The **CommandType** property must be set to **adCmdText** for use with OLE DB Provider for AS/400 and VSAM.

You can also use the **Command** object to open a data file after a **Connection** object has been opened and the **ActiveConnection** property has been set to this open connection. The **CommandText** property defines the data file to open (an **EXEC OPEN *DataSetName***statement, for example, where *DataSetName* represents a valid data file or library member on the host). You must set the **CommandType** property to **adCmdText** for use with OLE DB Provider for AS/400 and VSAM. If you open a host data file from a **Command** object, the data file is opened as read-only. This results from the limitation that no argument or option is passed by ADO that supplies a parameter describing whether the data set should be opened as read-only or updateable.

Connection Object in the OLE DB Provider for AS/400 and VSAM (ADO)

The ActiveX Data Objects (ADO) **Connection** object represents an open connection to an OLE DB data source. The **Provider** property sets the OLE DB provider. You can configure the connection before opening the data source by setting the **ConnectionString** properties. The **Version** property determines the version of the ADO implementation in use.

The **Open** method establishes the physical connection to the data source and the **Close** method terminates the connection. If errors occur, you can examine them with the **Errors** collection.

The following table lists the methods, properties, and collections for the **Connection** object that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

Name	Comment
Close Method	Closes a connection to a data source.
Execute Method	Evaluates command text as a table name. (The only supported *Options* parameter for this method is **adCmd Table**.)
Open Method	Opens a connection to a data source and may optionally pass **ConnectionString** parameters with this meth od. (The only supported *Options* parameter for this method is **adCmdText**.)
OpenSchema Me thod	Obtains database schema information from the OLE DB provider.
Attributes Metho d	Indicates one or more characteristics supported for a given **Connection** object.
ConnectionString Property	Contains the information used to establish a connection to a data source. (For more information, see the not e that follows.)
CursorLocation property	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).
Mode Property	Indicates the available permissions for modifying data in a connection.
Provider Propert y	Sets or returns the name of the provider for a connection.
State Property	Describes the current state of an object.
Version Property	Returns the version number of the ADO implementation in use.
Errors collection	Collections of **Error** objects on the connection.
Properties collec tion	Collections of properties on the connection.

Note that the information needed to establish a connection to a data source can be set in the **ConnectionString** property or passed as part of the **Open** method. In either case, this information must be in a specific format for use with OLE DB Provider for AS/400 and VSAM or OLE DB Provider for DB2. This information is either a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons.

ADO supports several standard ADO-defined arguments for the **ConnectionString** property, as listed in the following table.

Argu ment	Description
Data Sourc e	Name of the data source for the connection. This argument is optional when using OLE DB Provider for AS/400 and VS AM.

File Name	Name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed and is not supported by OLE DB Provider for AS/400 and VSAM.
Location	The remote database name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.
Password	Valid mainframe or AS/400 password to use when opening the connection. Host Integration Server 2009 uses this password to validate that the user can log on to the target host system and has appropriate access rights to the file.
Provider	Name of the provider to use for the connection. To use OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB." To use OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB."
User ID	Valid mainframe or AS/400 user name to use when opening the connection. Host Integration Server 2009 uses this user name to validate that the user can log on to the target host system and has appropriate access rights to the file.

OLE DB Provider for AS/400 and VSAM also supports a number of provider-specific arguments, some of which default to values in the registry. The following table lists these arguments.

Argument	**Description**
BinAsChar	This parameter indicates whether to process binary fields as character fields. (The default is 0; do not process binary fields as character fields.) This parameter is equivalent to the DBPROP_SNAOLEDB_BINASCHAR OLE DB property ID.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If you omit this argument, the default value is U.S./Canada (37). This parameter is equivalent to the DBPROP_SNAOLEDB_HOSTCCSID OLE DB property ID.
Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files. This parameter is equivalent to the DBPROP_SNAOLEDB_LIBRARY OLE DB property ID.
HCDFileName	The fully qualified file name of the distributed data management (DDM) Host Column Description (HCD) file. This parameter can be a UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the Host Integration Server system directory. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400. This parameter is equivalent to the DBPROP_SNAOLEDB_HCDPATH OLE DB property ID.
LocalLU	The name of the local logical unit (LU) alias configured in the SNA server. This parameter is equivalent to the DBPROP_SNAOLEDB_LOCALLU OLE DB property ID.
ModeName	The Advanced Program-to-Program Communications (APPC) mode must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes. This parameter is equivalent to the DBPROP_SNAOLEDB_APPCMODE OLE DB property ID.
NetAddr	When you select TCP/IP for the Network Transport Library (NTL), this parameter indicates the IP address of the host. This parameter is equivalent to the DBPROP_SNAOLEDB_NETADDRESS OLE DB property ID.
NetPort	When you select TCP/IP for the NTL, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446. This parameter is equivalent to the DBPROP_SNAOLEDB_NETPORT OLE DB property ID.

NetLib	This parameter determines whether you use TCP/IP or SNA APPC for network communication. The possible values for this parameter are TCPIP or SNA. The default value is SNA. This parameter is equivalent to the DBPROP_SNAOLEDB_NETTYPE OLE DB property ID.
PCCodePage	The character code page to use on the computer. If you omit this argument, the default value is set to Latin 1 (1252). This parameter is equivalent to the DBPROP_SNAOLEDB_PCCODEPAGE OLE DB property ID.
RDB	The remote database name for OS/400. You only need to specify this value if it is different from the remote LU alias configured in the SNA server.
RepairHostKeys	This parameter indicates whether the OLE DB provider should repair any host key values set in the registry, The default is **false**. This parameter is equivalent to the DBPROP_SNAOLEDB_REPAIRKEY OLE DB property ID.
RemoteLU	The name of the remote logical unit (LU) alias configured in the Host Integration Server computer. This parameter is equivalent to the DBPROP_SNAOLEDB_REMOTELU OLE DB property ID.
StrictVal	This parameter indicates whether strict validation should be used. The default is **false**. This parameter is equivalent to the DBPROP_SNAOLEDB_STRICTVAL OLE DB property ID.

A sample **ConnectionString** for use with OLE DB Provider for AS/400 and VSAM is as follows:

```
Conn.Provider="SNAOLEDB"
Conn.ConnectionString = "User ID=USERNAME;Password=password",&_
  "LocalLU=LOCAL;RemoteLU=DATABASE",&_
  "ModeName=QPCSUPP;CCSID=37;PCCodePage=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic®.

When opening a connection object in ADO 2.0, you must specify the **Prompt** connection property. For example, the following is valid with ADO 1.5 and ADO 2.0 and prompts the user for **ConnectionString** properties:

```
Conn.ConnectionString = "Provider=SNAOLEDB
Conn.Properties("PROMPT")=adPromptAlways
Conn.Open
```

A sample **Open** method call with these parameters is as follows:

```
RS.Open "library/member",Conn,2,1,1
```

The last three parameters to the **Open** method correspond with the *CursorType* (for example, the **adOpenDynamic** enum is 2), *LockType* (for example, the **adLockReadOnly** enum is 1), and *Options* (**adCmdText** is 1, which indicates that the source name should be evaluated as a table name). The *Options* parameter must be set to **adCmdText** (1) when used with a data source name with OLE DB Provider for AS/400 and VSAM.

The allowable values for CCSID when using SNA National Language Support (SNANLS) for character code conversions (the default) are listed in the following table.

EBCDIC character set	CCSID value
Arabic	20420
Binary (No Conversion)	65535

Chinese (Simplified)	935
Chinese (Traditional)	937
Cyrillic (Russian)	20880
Cyrillic (Serbian, Bulgarian)	21025
Denmark/Norway (Euro)	1142
Denmark/Norway	20277
Finland/Sweden (Euro)	1143
Finland/Sweden	20278
France (Euro)	1147
France	20297
Germany (Euro)	1141
Germany	20273
Greek (Modern)	875
Greek	20423
Hebrew	20424
Icelandic (Euro)	1149
Icelandic	20871
International (Euro)	1148
International	500
Italy (Euro)	1144
Italy	20280
Japanese (English-lower)	931
Japanese (Extend English)	939
Japanese (Extend Katakana)	930
Japanese (Katakana)	290
Japanese (Katakana-Kanji)	5026
Japanese (Latin-Kanji)	5035
Korean	933
Latin America/Spain (Euro)	1145
Latin America/Spain	20284
Latin-1 Open System (Euro)	20924
Latin-1 Open System	1047
Multilingual/ROECE (Latin-2)	870

Thai	20838
Turkish (Latin-5)	1026
Turkish	20905
U.S./Canada (Euro)	1140
U.S./Canada	37
United Kingdom (Euro)	1146
United Kingdom	20285

Note

SNANLS conversion uses the locale configured for the data sources using data links. For more information on SNANLS, see SNA National Language Support Programmer's Guide.

Error Object in the OLE DB Provider for AS/400 and VSAM (ADO)

The ActiveX Data Objects (ADO) **Error** object contains details about data access errors pertaining to a single operation involving ADO. You can read the properties of an **Error** object to obtain specific details about each error.

The **Error** object does not support any methods or collections. However, the **Errors** collection supported by other objects provides the standard **Collection** methods (**Clear** and **Delete**). OLE DB Provider automatically appends **Error** objects to the **Errors** collection when they occur.

The following table lists the **Error** object properties that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

Property Name	**Comment**
Description Property	The text of the error alert that is returned based on the minor error code (specific to OLE DB Provider for AS/400 and VSAM) contained in the **Error** object resulting from an error.
NativeError Property	A **Long** integer value of the error code returned by OLE DB Provider for AS/400 and VSAM.
Number Property	The **Long** integer value of the error constant.
Source Property	A **string** that indicates the name of the object or application that originally generated an error.

Field Object in the OLE DB Provider for AS/400 and VSAM (ADO)

The ActiveX Data Objects (ADO) **Field** object represents a column of data with a common data type. Each **Field** object corresponds to a column in a **Recordset** object.

The following table lists the **Field** object methods, properties, and collections that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

Name	Comment
AppendChunk Method	Appends data to a large text or binary data **Field** object.
GetChunk Method	Returns all or portions of the contents of a large text or binary data **Field** object.
ActualSize Property	Actual length of the value of a field.
Attributes Property	One or more characteristics supported for a given **Field** object.
DefinedSize Property	Defined size of a **Field** object.
Name Property	Name of the **Field** object.
NumericScale Property	Scale of numeric values in a **Field** object for numeric data.
OriginalValue Property	Value of a **Field** object that existed in the record before changes were made.
Precision Property	Degree of precision for numeric values in a **Field** object for numeric data.
Type Property	Operational type or data type for a **Field** object.
UnderlyingValue Property	Current value of a **Field** object.
Value Property	Value assigned to a **Field** object in a **Recordset**.
Properties collection	Collections of properties on the field.

Recordset Object in the OLE DB Provider for AS/400 and VSAM (ADO)

The ActiveX Data Objects (ADO) **Recordset** object represents the entire set of records from a base table. At any time, the **Recordset** object refers to only one record within the set as the current record.

The following table lists the **Recordset** object methods, properties, and collections that the current version of Microsoft OLE DB Provider for AS/400 and VSAM supports.

Name	Comment
AddNew Method	Creates a new record for an updateable **Recordset** object.
CancelBatch Method	Cancels a pending batch update.
CancelUpdate Method	Cancels any changes made to a current record or to a new record prior to calling the **UpdateBatch** method.
Clone Method	Creates a duplicate **Recordset** object from an existing **Recordset** object.
Close Method	Closes an open object and any dependent objects.
Delete Method	Deletes the current record in an open **Recordset** object or an object from a collection.
Find Method	Finds the next record to match a condition (ADO 1.5 and later).
GetRows Method	Retrieves multiple records of a **Recordset** into an array.
Move Method	Moves the position of the current record in a **Recordset** object.
MoveFirst Method	Moves to the first record in a specified **Recordset**.
MoveLast Method	Moves to the last record in a specified **Recordset**.
MoveNext Method	Moves to the next record in a specified **Recordset**.
MovePrevious Method	Moves to the previous record in a specified **Recordset**.
Open Method	Opens a cursor on a **Recordset**.
Requery Method	Updates the data in a **Recordset** object by re-executing the query on which the object is based (equivalent to calling the **Close** and **Open** methods in succession).
Save Method	Saves a **Recordset** in a file or **Stream** object.
Seek method	
Supports Method	Determines whether a specified **Recordset** object supports a particular type of function.
Update Method	Saves any changes you make to the current record of a **Recordset** object.
UpdateBatch Method	Writes all pending batch updates to disk.
ActiveCommand Property	Returns the **Command** object that created the specified **Recordset**.

ActiveConnection Property	Sets or returns the **Connection** object that the specified **Recordset** object currently belongs.
BOF Property	Indicates whether the current record position is before the first record in a **Recordset** object.
Bookmark Property	Returns a bookmark that uniquely identifies the current record in a **Recordset** object or sets the current record in a **Recordset** object identified by a valid bookmark.
CacheSize Property	Sets or returns the number of records from a **Recordset** object that are cached locally in memory.
CursorLocation Property	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).
CursorType Property	Sets or returns the type of cursor used in a **Recordset** object. Only the **adOpenDynamic CursorType** is supported by the current version of OLE DB Provider for AS/400 and VSAM.
EditMode Property	Indicates the editing status of the current record type.
EOF Property	Indicates whether the current record position is after the last record in a **Recordset** object.
Filter Property	Indicates a filter for data in a **Recordset** (revised in ADO 1.5 and later).
LockType Property	Sets or returns the types of locks placed on records during editing. All four lock types (**adLockReadOnly**, **adLockOptimistic**, **adLockPessimistic**,and **adLockBatchOptimistic**) are supported by the OLE DB Provider for AS/400 and VSAM. Note that the OLE DB provider internally maps **adLockPessimistic** to a LockType of **adLockBatchOptimistic**.
Sort Property	Indicates the column names and order to sort data in a **Recordset** object (new property in ADO 1.5 and later).
Source Property	Sets or returns the source (table name or command object) for the data in a **Recordset**.
State Property	Describes the current state of an object.
Status Property	Indicates the status of the current record with respect to batch updates or other bulk operations.
Fields collection	Collections of fields on the **Recordset**.
Properties collection	Collections of properties on the **Recordset**.

The syntax supported by the OLE DB Provider for AS/400 and VSAM to open a recordset (table) using the **Recordset.Open** method is as follows:

```
EXEC OPEN TableName
```

where *TableName* represents one of the host file naming conventions listed in the following table.

Host file type	File naming convention
VSAM Data Sets	DATASETNAME.FILENAME
Partitioned Data Sets	DATASETNAME.FILENAME(MEMBER)
OS/400 Files	LIBRARY/FILE
OS/400 Files	LIBRARY/FILE.NAME
OS/400 File Members	LIBRARY/FILE(MEMBER)
OS/400 File Members	LIBRARY/FILE.NAME(MEMBER)

Note that if a member of a library contains a dot in the member name, you must surround the member name with double

quotes. For example, if the member name is NAMES.DAT, the proper syntax for command text used for the **Recordset.Open** method is:

```
RecordSet.Open "EXEC OPEN LIBRARY/FILE(""NAMES.DAT"")",...
```

You must specify the full path to the mainframe data set. In the preceding example, there are two path elements (LIBRARY/FILE) and one name element (NAMES.DAT).

Whenever a data set is allocated, it is given a unique name composed of one or more segments. Each segment of the data set name is joined by periods and represents a level of qualification. For example, the following data set has four segments that comprise the fully qualified data set name (three path elements and one name element):

```
SAMPLES.DEMO.KSDS.TITLES
```

The high-level qualifier is SAMPLES. The low-level qualifier is TITLES. Each segment can be from 1 through 8 characters in length. (The first character must be alphabetical, and the remainder can be alphanumeric or hyphens.) The full data set name must be no more than 44 characters in length and contain no more than 22 segments.

The Recordset **Bookmark** method is supported for all AS/400 physical and logical files, as well as the following mainframe file types:

- KSDS if the file has a unique key
- RRDS if the file has a unique key

You can use the Recordset **AddNew** method on Entry-Sequenced Data Sets (ESDS) files on the AS/400 only when you are positioned at the end of the **Recordset** object (file). With Alternate Index files on the AS/400, you can use the **AddNew** method to add records when at the end of the **Recordset** object or by key. With Key-Sequenced Data Sets (KSDS) or Fixed-Length Relative Record Data Sets (RRDS) files on the mainframe, the **AddNew** method adds new records by key.

To use the Recordset **Find** method or the **Filter** property, an AS/400 logical file, an AS/400 keyed physical file, a mainframe KSDS file with a unique key, or a mainframe RRDS file with a unique key must be used. If you use these methods or properties on an AS/400 non-keyed physical file or any other mainframe file type, the method fails.

The Recordset **Sort** property is used with an open **Recordset** object based on an AS/400 physical file. The **Sort** property enables the user to indicate which logical view to apply to an AS/400 physical file. The logical view must be a valid index specified in the description of the AS/400 physical file. The AS/400 logical file provides the logical view. OLE DB Provider for AS/400 and VSAM responds to the **Sort** property request by first closing the open physical file, and then opening the logical file that points back to the data in the physical file.

The **Recordset Sort** property is only supported on AS/400 hosts. If the user opens a **Recordset** object based on an AS/400 logical file, there is probably no need to use **Recordset**.**Sort**. For performance reasons, applications should be written to open the AS/400 logical file first, because the overhead is so much greater when opening a physical file first.

ADO Object Support in the OLE DB Provider for DB2

The following table summarizes the Microsoft® ActiveX® Data Objects (ADO) version 2.0 objects that the current version of Microsoft OLE DB Provider for DB2 supports.

ADO object	Support
Collection	Yes, most methods
Command Object	Yes, some methods, some properties, and all collections
Connection Object	Yes, some methods, some properties, and all collections
Error Object	Yes, some properties
Field Object	Yes, no methods, most properties, and all collections
Parameter Object	Yes, most methods, most properties, and all collections
Recordset Object	Yes, most methods, most properties, and all collections

This section contains:

- ADO Method Support in the OLE DB Provider for DB2
- ADO Property Support in the OLE DB Provider for DB2
- ADO Collection Support in the OLE DB Provider for DB2
- Command Object in the OLE DB Provider for DB2 (ADO)
- Connection Object in the OLE DB Provider for DB2 (ADO)
- Error Object in the OLE DB Provider for DB2 (ADO)
- Field Object in the OLE DB Provider for DB2 (ADO)
- Recordset Object in the OLE DB Provider for DB2 (ADO)

ADO Method Support in the OLE DB Provider for DB2

The following table summarizes the ActiveX Data Objects (ADO) version 2.0 object methods that the current version of Microsoft OLE DB Provider for DB2 supports.

ADO object	Method	Support
Collection	**Append**	No
	Clear Method	Yes
	Delete Method	Yes
	Item Method	Yes
	Refresh Method	Yes
Command Object	**CreateParameter** Method	Yes
	Cancel Method	No
	Execute Method	Yes
Connection Object	**BeginTrans** Method	Yes
	Cancel Method	No
	Close Method	Yes
	CommitTrans Method	Yes
	Execute Method	Yes
	Open Method	Yes
	OpenSchema Method	Yes
	RollbackTrans Method	Yes
Field Object	**AppendChunk** Method	No
	GetChunk Method	No
	ReadFromFile Method	No
	WriteToFile Method	No
Parameter Object	**AppendChunk** Method	No
Recordset Object	AddNew Method	Yes
	Cancel Method	No
	CancelBatch Method	Yes
	CancelUpdate Method	Yes
	Clone Method	Yes
	Close Method	Yes
	Delete Method	Yes
	Find Method	No

	GetRows Method	Yes
	Move Method	Yes
	MoveFirst Method	Yes
	MoveLast Method	No
	MoveNext Method	Yes
	MovePrevious Method	No
	NextRecordset Method	No
	Open Method	Yes
	Requery Method	Yes
	Resync Method	No
	Save Method	Yes
	Seek Method	No
	Supports Method	Yes
	Update Method	Yes
	UpdateBatch Method	Yes

Note

The **Collection** object is a special case, representing a collection of other ADO objects. These collection objects support several methods:

- **Append** to add an object to a collection
- **Clear** to empty all objects from a collection
- **Delete** to remove a single object from a collection
- **Item** to return a specific member object of a collection by name or ordinal number
- **Refresh** to update the objects in a collection to reflect objects available from and specific to the OLE DB provider

ADO Property Support in the OLE DB Provider for DB2

The following table summarizes the ActiveX Data Objects (ADO) version 2.0 object properties that the current version of Microsoft OLE DB Provider for DB2 supports.

ADO object	Property	Support
Command Object	ActiveConnection Property	Yes
	CommandText Property	Yes
	CommandTimeout Property	No
	CommandType Property	Yes
	Prepared Property	Yes
	State Property	Yes
Connection Object	Attributes Property	Yes
	CommandTimeout Property	No
	ConnectionString Property	Yes
	ConnectionTimeout Property	No
	CursorLocation Property	Yes
	DefaultDatabase Property	No
	IsolationLevel Property	Yes
	Mode Property	Yes
	Provider Property	Yes
	State Property	Yes
	Version Property	Yes
Error Object	Description Property	Yes
	HelpContext Property	No
	HelpFile Property	No
	NativeError Property	Yes
	Number Property	Yes
	Source Property	Yes
	SQLState Property	Yes
Field Object	ActualSize Property	Yes
	Attributes Property	Yes
	DataFormat Property	No
	DefinedSize Property	Yes
	Name Property	Yes

	NumericScale Property	Yes
	OriginalValue Property	Yes
	Precision Property	Yes
	Type Property	Yes
	UnderlyingValue Property	Yes
	Value Property	Yes
Parameter Object	**Attributes** Property	Yes
	Direction Property	Yes
	Name Property	Yes
	NumericScale Property	Yes
	Precision Property	Yes
	Size Property	Yes
	Type Property	Yes
	Value Property	Yes
Recordset Object	**AbsolutePage** Property	No
	AbsolutePosition Property	No
	ActiveCommand Property	Yes
	ActiveConnection Property	Yes
	BOF Property	Yes
	Bookmark Property	Yes
	CacheSize Property	Yes
	CursorLocation Property	Yes
	CursorType Property	Yes
	DataMember Property	No
	DataSource Property	No
	EditMode Property	Yes
	EOF Property	Yes
	Filter Property	No
	Index Property	No
	LockType Property	Yes
	MarshalOptions Property	No
	MaxRecords Property	Yes
	PageCount Property	No

	PageSize Property	No
	RecordCount Property	No
	Sort Property	No
	Source Property	Yes
	State Property	Yes
	Status Property	Yes
	StayInSync Property	No

ADO Collection Support in the OLE DB Provider for DB2

The following table summarizes the ActiveX Data Objects (ADO) version 2.0 object collections that Microsoft OLE DB Provider for DB2 supports.

ADO object	Collection	Support
Command Object	**Parameters** Collection	Yes
	Properties Collection	Yes
Connection Object	**Errors** Collection	Yes
	Properties Collection	Yes
Field Object	**Properties** Collection	Yes
Parameter Object	**Properties** Collection	Yes
Recordset Object	**Fields** Collection	Yes
	Properties Collection	Yes

Command Object in the OLE DB Provider for DB2 (ADO)

The ActiveX Data Objects (ADO) **Command** object is a definition of a specific command that is to be executed against an OLE DB data source.

You can use **Command** objects to create a **Recordset** object and obtain records, to execute a bulk operation, or to manipulate the structure of a database. When using Microsoft OLE DB Provider for DB2, some collections, methods, or properties of a **Command** object may generate an error when called.

The primary purpose of the **Command** object in the context of OLE DB Provider for DB2 is to issue SQL commands for execution by the remote DB2 target server. Legal SQL commands are documented for the target DB2 platforms in SQL Reference Guides published by IBM.

The following table lists the **Command** object methods, properties, and collections that the current version of OLE DB Provider for DB2 supports.

Name	Comment
Execute Method	Evaluates command text (only supported *Options* parameter for this method is **adCmdText**, which indicates that this is an SQL text command).
ActiveConnection Property	Sets or returns the information used to establish a connection to a data source. (For more information, see the notes that follow.)
CommandText Property	Sets or returns the command text to be executed.
CommandType Property	Sets or returns the type of command in a **CommandText** property.
State Property	Describes the current state of an object.
Properties collection	Collections of properties on the command.

The **Execute** method executes a command and returns a **Recordset** object, if appropriate. You can use the **Command** object to open tables or execute SQL commands on a remote DB2 server. If errors occur, you can examine these with the **Errors** collection on the **Connection** object.

You can create a **Command** object independently of a previously defined **Connection** object by setting the **ActiveConnection** property of the **Command** object to a valid connection string. (For the proper syntax, see the **ConnectionString** property of the **Connection** object.) ADO still creates a **Connection** object, but it does not assign that object to an object variable. However, if multiple **Command** objects are to be associated with the same connection, the **Connection** object needs to be explicitly created and opened. This assigns the **Connection** object to an object variable. If the **ActiveConnection** property of the **Command** object is not set to this object variable, ADO creates a new **Connection** object for each **Command** object, even if the same connection string is used.

The **ActiveConnection** property associates an open connection with a **Command** object. The **CommandText** property defines the text version of a command (for example, **SELECT ALL FROM TABLE**). The **CommandType** property specifies the type of command described in the **CommandText** property prior to execution to optimize performance. The **CommandType** property must be set to **adCmdText** for use with OLE DB Provider for DB2.

Connection Object in the OLE DB Provider for DB2 (ADO)

The ActiveX Data Objects (ADO) **Connection** object represents an open connection to an OLE DB data source. The **Provider** property sets the OLE DB provider to use. The connection can be configured before opening the data source by setting the **ConnectionString** properties. The version of the ADO implementation in use can be determined from the **Version** property.

The physical connection to the data source is established using the **Open** method and terminated with the **Close** method. If errors occur, these can be examined with the **Errors** collection.

The following table lists the **Connection** object methods, properties, and collections that the current version of Microsoft OLE DB Provider for DB2 supports.

Name	Comment
Close Method	Closes a connection to a data source.
Execute Method	Evaluates command text.
Open Method	Opens a connection to a data source and may optionally pass **ConnectionString** parameters with this method.
OpenSchema Method	Obtains database schema information from the OLE DB provider.
Attributes Property	One or more characteristics supported for a given **Connection** object.
ConnectionString Property	Contains the information used to establish a connection to a data source. (For more information, see the notes that follow.)
CursorLocation Property	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).
IsolationLevel Property	Sets or returns the level of isolation for a **Connection** object.
Mode Property	Indicates the available permissions for modifying data in a connection.
Provider Property	Sets or returns the name of the provider for a connection.
State Property	Describes the current state of an object.
Version Property	Returns the version number of the ADO implementation in use.
Errors collection	Collections of **Error** objects on the connection.
Properties collection	Collections of properties on the connection.

You can set the information needed to establish a connection to a data source in the **ConnectionString** property or pass it as part of the **Open** method. In either case, this information must be in a specific format for use with OLE DB Provider for DB2. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons. ADO supports several standard ADO-defined arguments for the **ConnectionString** property as listed in the following table.

Argument	Description
Data Source	Name of the data source for the connection. This argument is optional when using OLE DB Provider for DB2.
File Name	Name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed.
Location	The remote database name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.

Password	Valid mainframe or AS/400 password to use when opening the connection. This password is used to verify that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is equivalent to the DBPROP_AUTH_PASSWORD OLE DB property ID.
Provider	Name of the provider to use for the connection. To use OLE DB Provider for DB2, you must set the Provider string to "DB2OLEDB."
User ID	Valid mainframe or AS/400 user name to use when opening the connection. This user name validates that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is equivalent to the DBPROP_AUTH_USERID OLE DB property ID.

The OLE DB Provider for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the following tables.

The following table lists the arguments supported by the OLE DB Provider for DB2 supplied with Host Integration Server 2009.

Argument	**Description**
BinAsChar	When this parameter is set to **true**, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This parameter defaults to **false**.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If you omit this argument, the default value is U.S./Canada (37).
DefSch	The name of the default schema (collection/owner) where the system catalogs resides. This parameter can be QSYS2, SYSIBM, SYSTEM, CURLIB, or USERID depending on platform. This parameter does not have a default value.
InitCat	You use this parameter as the first part of a three-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. In DB2/400, this parameter is referred to as RDBNAM. The RDBNAM value can be determined by invoking the **WRKRDBDIRE** command from the console to the OS/400 system. If there is no RDBNAM value, you can create one using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE. This parameter has no default value.
LocalLU	The name of the local logical unit (LU) alias configured in Host Integration Server.
ModeName	The Advanced Program-to-Program Communications (APPC) mode must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.

NetAddr	When you select TCP/IP for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When you select TCP/IP for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether you use TCP/IP or SNA APPC for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the computer. If you omit this argument, the default value is set to Latin 1 (1252).
PkgCol	The name of the Distributed Relational Database Architecture (DRDA) target collection (AS/400 library) where Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. Microsoft OLE DB Provider for DB2 uses packages to issue dynamic and static SQL statements. The OLE DB provider creates packages dynamically in the location to which the user points using the Package Collection parameter.
RemoteLU	The name of the remote LU alias configured in Host Integration Server.
TPName	The transaction program name when used with SQL/DS.
UOW	This parameter determines whether two-phase commit is enabled. The possible values for this parameter are distributed unit of work (DUW) or remote unit of work (RUW). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** using OLE DB Provider for DB2 is as follows:

```
Conn.Provider="DB2OLEDB"
Conn.ConnectionString = "User ID=USERNAME;Password=password",&_
  "LocalLU=LOCAL;RemoteLU=DATABASE",&_
```

```
    "ModeName=QPCSUPP;CCSID=37;PcCodePage=437"
  Conn.Properties("PROMPT")=adPromptNever
  Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic®.

When opening a connection object in ADO 2.0, you must specify the **Prompt** connection property. For example, the following is valid with ADO 1.5 and ADO 2.0 and will prompt the user for **ConnectionString** properties:

```
Conn.ConnectionString = "Provider=DB2OLEDB
Conn.Properties("PROMPT")=adPromptAlways
Conn.Open
```

A sample **Open** method call with these parameters is as follows:

```
RS.Open "Accounting",Conn,0,1,1
```

The last three parameters to the **Open** method correspond with the *CursorType* (for example, the **adOpenForwardOnly** enum is 0), *LockType* (for example, the **adLockReadOnly** enum is 1), and *Options* (**adCmdText** is 1, which indicates that the source name should be evaluated as SQL text). The *Options* parameter must be set to **adCmdText** (1) when used with a data source name with OLE DB Provider for DB2.

The allowable values for the Character Code Set Identifier (CCSID) when using SNA National Language Support (SNANLS) for character code conversions (the default) are listed in the following table.

EBCDIC character set	CCSID value
Arabic	20420
Binary (No Conversion)	65535
Chinese (Simplified)	935
Chinese (Traditional)	937
Cyrillic (Russian)	20880
Cyrillic (Serbian, Bulgarian)	21025
Denmark/Norway (Euro)	1142
Denmark/Norway	20277
Finland/Sweden (Euro)	1143
Finland/Sweden	20278
France (Euro)	1147
France	20297
Germany (Euro)	1141
Germany	20273
Greek (Modern)	875
Greek	20423

Hebrew	20424
Icelandic (Euro)	1149
Icelandic	20871
International (Euro)	1148
International	500
Italy (Euro)	1144
Italy	20280
Japanese (English-lower)	931
Japanese (Extend English)	939
Japanese (Extend Katakana)	930
Japanese (Katakana)	290
Japanese (Katakana-Kanji)	5026
Japanese (Latin-Kanji)	5035
Korean	933
Latin America/Spain (Euro)	1145
Latin America/Spain	20284
Latin-1 Open System (Euro)	20924
Latin-1 Open System	1047
Multilingual/ROECE (Latin-2)	870
Thai	20838
Turkish (Latin-5)	1026
Turkish	20905
U.S./Canada (Euro)	1140
U.S./Canada	37
United Kingdom (Euro)	1146
United Kingdom	20285

Note that the SNA National Language Support (SNANLS) conversion uses the locale configured for the data sources using data links. For more information, see the SDK documentation under the SNA National Language Support Programmer's Guide.

The allowable values for CCSID when using ANSI/OEM for character code conversions are listed in the following table.

ANSI/OEM character set	**CCSID value**
ANSI - Arabic	1256
ANSI - Baltic	1257
ANSI - Cyrillic	1251
ANSI - Eastern Europe	1250

ANSI - Greek	1253
ANSI - Hebrew	1255
ANSI - Latin I	1252
ANSI - Turkish	1254
ANSI/OEM - Korean (Extended Wansung)	949
ANSI/OEM - Japanese Shift-JIS	932
ANSI/OEM - Simplified Chinese GBK	936
ANSI/OEM - Traditional Chinese Big5	950
ANSI/OEM - Thai	874
ANSI/OEM - Vietnam	1258

Error Object in the OLE DB Provider for DB2 (ADO)

The ActiveX Data Objects (ADO) **Error** object contains details about data access errors pertaining to a single operation involving ADO. You can read the properties of an **Error** object to obtain specific details about each error.

The **Error** object does not support any methods or collections. However, the **Errors** collection supported by other objects provides the standard **Collection** methods (**Clear** and **Delete**). The OLE DB provider automatically appends **Error** objects to the **Errors** collection when they occur.

The following table lists the **Error** object properties that the current version of Microsoft OLE DB Provider for DB2 supports.

Property Name	**Comment**
Description Property	The text of the error alert that is returned based on the minor error code (specific to the OLE DB Provider for DB2) contained in the **Error** object resulting from an error.
NativeError Property	A **Long** integer value of the error code returned by the OLE DB Provider for DB2.
Number Property	The **Long** integer value of the OLE DB error constant.
Source Property	A string that indicates the name of the object or application that originally generated an error.

Field Object in the OLE DB Provider for DB2 (ADO)

The ActiveX Data Objects (ADO) **Field** object represents a column of data with a common data type. Each **Field** object corresponds to a column in a **Recordset** object.

The following table lists the **Field** object methods, properties, and collections that the current version of Microsoft OLE DB Provider for DB2 supports.

Name	Comment
ActualSize Property	Actual length of a field's value.
Attributes Property	One or more characteristics supported for a given **Field** object.
DefinedSize Property	Defined size of a **Field** object.
Name Property	Name of the **Field** object.
NumericScale Property	Scale of numeric values in a **Field** object for numeric data.
OriginalValue Property	Value of a **Field** object that existed in the record before changes were made.
Precision Property	Degree of precision for numeric values in a **Field** object for numeric data.
Type Property	Operational type or data type for a **Field** object.
UnderlyingValue Property	Current value of a **Field** object.
Value Property	Value assigned to a **Field** object in a **Recordset**.
Properties collection	Collections of properties on the field.

Recordset Object in the OLE DB Provider for DB2 (ADO)

The ActiveX Data Objects (ADO) **Recordset** object represents the entire set of records from a base table. At any time, the **Recordset** object refers to only one record within the set as the current record.

The following table lists the **Recordset** object methods, properties, and collections that the current version of Microsoft OLE DB Provider for DB2 supports.

Name	Comment
AddNew Method	Creates a new record for an updateable **Recordset** object.
CancelBatch Method	Cancels a pending batch update.
CancelUpdate Method	Cancels any changes made to a current record or to a new record prior to calling the **UpdateBatch** method.
Clone Method	Creates a duplicate **Recordset** object from an existing **Recordset** object.
Close Method	Closes an open object and any dependent objects.
Delete Method	Deletes the current record in an open **Recordset** object or an object from a collection.
GetRows Method	Retrieves multiple records of a **Recordset** into an array.
Move Method	Moves the position of the current record in a **Recordset** object.
MoveFirst Method	Moves to the first record in a specified **Recordset**.
MoveNext Method	Moves to the next record in a specified **Recordset**.
Open Method	Opens a cursor on a **Recordset**.
Requery Method	Updates the data in a **Recordset** object by re-executing the query on which the object is based (equivalent to calling the **Close** and **Open** methods in succession).
Save Method	Saves a **Recordset** in a file or **Stream** object.
Supports Method	Determines whether a specified **Recordset** object supports a particular type of function.
Update Method	Saves any changes you make to the current record of a **Recordset** object.
UpdateBatch Method	Writes all pending batch updates to disk.
ActiveCommand Property	Returns the **Command** object that created the specified **Recordset**.
ActiveConnection Property	Sets or returns the **Connection** object that the specified **Recordset** object currently belongs.
BOF Property	Indicates whether the current record position is before the first record in a **Recordset** object.
Bookmark Property	Returns a bookmark that uniquely identifies the current record in a **Recordset** object or sets the current record in a **Recordset** object identified by a valid bookmark.
CacheSize Property	Sets or returns the number of records from a **Recordset** object that are cached locally in memory.
CursorLocation Property	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).

CursorType Property	Sets or returns the type of cursor used in a **Recordset** object. The current version of OLE DB Provider for DB2 supports only the **adOpenForwardOnly CursorType**.
EditMode Property	Indicates the editing status of the current record type.
EOF Property	Indicates whether the current record position is after the last record in a **Recordset** object.
LockType Property	Sets or returns the types of locks placed on records during editing. OLE DB Provider for DB2 supports locks of type **adLockReadOnly** and **adLockPessimistic**.
MaxRecords Property	Sets or returns the maximum number of records to return to a **Recordset** from a query.
Source Property	Sets or returns the source (table name or command object) for the data in a **Recordset**.
State Property	Describes the current state of an object.
Status Property	Indicates the status of the current record with respect to batch updates or other bulk operations.
Fields collection	Collections of fields on the **Recordset**.
Properties collection	Collections of properties on the **Recordset**.

ODBC Driver for DB2 Programmer's Guide

The Microsoft ODBC Driver for DB2 enables users to access IBM DB2 DB2 from within an ODBC-aware application. ODBC defines a standard set of interfaces that provide access to disparate databases. The ODBC Driver for DB2 combines the data access of ODBC with the underlying Microsoft Distributed Relational Database Architecture (DRDA) application requester also used by the Microsoft OLE DB Provider for DB2. Using this combination of technologies, the ODBC Driver for DB2 can provide database access to IBM's Distributed Relational Database Architecture and IBM DB2.

Organizations have invested in secure, robust, enterprise-wide data storage and management systems. DRDA is a set of rules for distributing or extending relational data from one computer to another, such as a server computer to an IBM DB2 database server running on a mainframe or an AS/400 computer. By combining the ODBC and DRDA architectures, Microsoft allows organizations to preserve their investments in an existing data management infrastructure, while extending data access to all enterprise-wide DB2 data sources.

The ODBC Driver for DB2 can be used interactively or from an application program to issue SQL statements and execute DB2 stored procedures. From Microsoft Excel, users can import DB2 tables into worksheets and use Excel graphing tools to analyze the data. From Microsoft Access, users can import from and export to DB2. With Microsoft Internet Information Services (IIS), developers can publish DB2-stored information to users through a Web browser.

For API reference and other technical information about the ODBC Driver for DB2, see the ODBC Driver for DB2 Programmer's Reference section of the SDK.

For more information about how to use the ODBC Driver for DB2, see the ODBC Driver for DB2 section in the Operations guide.

In This Section

Goals of the ODBC Driver for DB2

Relational database management systems (RDBMS) are one of the major sources of mission-critical information in today's enterprise organizations. Relational database technology enables departments and individual users to save their information in centrally managed database stores that can be maintained by the organization's information systems group.

IBM DB2 is a popular RDBMS for a significant number of enterprise customers. Customers need a cost-effective and manageable means to integrate DB2 with Microsoft® SQL Server™, Microsoft Internet Information Services (IIS), and Microsoft Office applications. The goal of Microsoft ODBC Driver for DB2 is to provide customers and solution providers with the means to integrate desktop database applications with this wealth of data residing on IBM DB2 database systems.

ODBC Driver for DB2 Architecture

The Microsoft ODBC Driver for DB2 is an ODBC-compliant database driver for Microsoft Windows Server 2003 and Windows 2000 that enables your existing ODBC applications access data residing in IBM DB2 database servers without changing any code. The ODBC Driver for DB2 can connect ODBC-compliant applications with DB2 data sources using the underlying Microsoft Distributed Relational Database Architecture (DRDA) application requester. The ODBC application connects to the ODBC Driver for DB2. These ODBC requests are processed by the underlying Microsoft DRDA application requester. The data is then passed by an SQL interface to the DB2 data store.

The ODBC Driver for DB2 shares the same DRDA application requester that is used by the Microsoft OLE DB Provider for DB2. The DRDA application requester is the network client that provides remote database access to DB2 across an SNA LU 6.2 and TCP/IP network.

The ODBC Driver for DB2 is compliant with the Microsoft Open Database Connectivity (ODBC) specification. ODBC is a specification for an application program interface (API) that enables applications to access multiple database systems using SQL.

See Also

Other Resources

ODBC Driver for DB2 Programmer's Guide

Platforms Supported by the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2 supports popular DB2 platforms supported by the Microsoft OLE DB Provider for DB2 because both use the same underlying Distributed Relational Database Architecture (DRDA) application requester.

The ODBC Driver for DB2 offers network connectivity using SNA APPC LU 6.2 connectivity, as well as native TCP/IP (not reliant on any special IBM or third-party routers).

See Also

Other Resources

ODBC Driver for DB2 Programmer's Guide

ODBC Driver for DB2 Requirements

Information about hardware and software requirements for ODBC Driver for DB2 can be found in Host Integration Server 2009.

When connecting over SNA using LU 6.2, the ODBC Driver for DB2 requires the following computer-to-host connectivity software:

- Microsoft Host Integration Server 2009
- Microsoft Host Integration Server Client

Note that the ODBC Driver for DB2 does not require any special host connectivity software when connecting directly to a host system using TCP/IP.

The ODBC Driver for DB2 supports the following OLE DB and ADO versions:

The ODBC Driver for DB2 supplied with Host Integration Server supports the following ADO version:

- ADO version 2.5. The Host Integration Server data access features require the runtime libraries for ADO version 2.5. On Windows Server 2003 and Windows 2000, these ADO libraries are installed as part of the Windows Server 2003 operating systems or Windows 2000.

Configuring ODBC Data Sources

A data source associates a particular ODBC driver with the data to be accessed through that driver. Data source information must be configured for each DB2 system that is to be accessed using the ODBC Driver for DB2. The default parameters for the ODBC Driver for DB2 are used for the data source only when these parameters are not configured for each data source.

An ODBC data source name (DSN) can be one of the following types:

- User. A data source local to a computer and accessible only by the current user that created the data source.
- System. A data source local to a computer but not dedicated to a specific user, so any user with appropriate privileges can access a system DSN. A System data source is visible to all users on a computer, including Windows NT services.
- File. A data source stored in a file that can be shared among all users who have the same ODBC drivers installed. These data sources need not be dedicated to a specific user or local to a computer.

User and System data sources are stored in the registry. File data sources are stored as files with a file extension of .dsn. File DSNs can be stored in any location on the file system including remotely mounted shares. By default, File DSNs are stored in the following location:

C:\Program Files\Common Files\ODBC\Data Sources

ODBC data sources can be configured using the **ODBC Data Source Administrator**. On Microsoft Windows Server 2003 and Windows 2000, a shortcut to the **ODBC Data Source Administrator** is located in the Control Panel under **Administrative Tools** as **Data Sources (ODBC)**. The Data Access Tool provided as part of the ODBC Driver for DB2 enables users to create and modify ODBC data sources. This tool makes calls to the ODBC Data Source Administrator application to provide these functions.

- Using the Microsoft ODBC Driver for DB2 Configuration Dialog Box
- Configuration Property Mappings Between the ODBC Driver for DB2 and the OLE DB Provider for DB2
- ODBC Connection String Attributes

Using the Microsoft ODBC Driver for DB2 Configuration Dialog Box

The **Microsoft ODBC Driver for DB2 Configuration** dialog box contains five tabs, which are described in the following topics.

This section contains:

- General
- Connection
- Security
- Target Database
- Locale

General

The **General** tab enables the user to configure the data source name required to connect to DB2. For the Microsoft ODBC Driver for DB2 supplied with Host Integration Server 2009, the **General** tab contains the following fields.

Parameter	Comments
Data Source Name	A blank field for specifying the name of the data source. Enter a string that identifies this ODBC data source. The data source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For file data sources, this field is used to name the DSN file, which is stored in C:\Program Files\Common Files\ODBC\Data Sources.
Description	A blank field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.

Connection

The **Connection** tab enables the user to configure the basic attributes required to connect to a data source. For the Microsoft® ODBC Driver for DB2, the **Connection** tab has the following fields.

Parameter	Comments
Network transport	An option button is used to select the network transport. Valid options are **APPC Connection (SNA LU 6.2)** or **TCP/IP Connection**. For the default, **APPC Connection**, the values for APPC local LU alias, APPC remote LU alias, and APPC mode name are required. For **TCP/IP Connection**, the values for IP address and Network port are required.
APPC local LU alias	When **APPC Connection** is selected, this field is the name of the local LU alias configured in Host Integration Server.
APPC remote LU alias	When **APPC Connection** is selected, this field is the name of the remote LU alias configured in Host Integration Server.
APPC mode name	When **APPC Connection** is selected, this field is the APPC mode and must be set to a value that matches the host configuration and SNA service configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). The default is QPCSUPP.
IP address	When **TCP/IP Connection** is selected as the network transport, this field indicates the IP address of the host DB2 server.
Network port	When **TCP/IP Connection** is selected as the network transport, this field indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default is IP port 446.

The **Connection** tab also includes a **Test connection** button that may be used to test the connection parameters. A connection can only be tested after all of the required parameters for the **Connection** tab and other ODBC data source parameters are configured properly. When this button is clicked, a session is established with the remote DB2 system using ODBC Driver for DB2

Security

The **Security** tab enables the user to configure optional attributes used to restrict connections to a data source.

For the Microsoft ODBC Driver for DB2 in Host Integration Server 2009, the **Security** tab has the following fields.

Parameter	**Comments**
Authentication	An option button is used to select the type of authentication. Valid options are **Use this username or Use single signon**. For the default **Use this username** option, the value for the user name is required.
Use this username	When this option is selected, authentication is based on the user name entered in the text box. A valid user name is normally required to access data on DB2. A user name can remain optionally in the DSN. The ODBC Driver for DB2 prompts the user at run time to enter a valid password. Additionally, the prompt dialog box enables the user to override the user name that is stored in the DSN.
Use Single Sign-On	An option button to select whether Single Sign-On (SSO) or a specific user name should be used. SSO is an optional Host Security feature. SSO enables the administrator to create data source definitions that isolate the logon process from the end user.

The AS/400 computer is case-sensitive with regard to user IDs and passwords. When connecting to DB2 for OS/400, user names and passwords must be in uppercase. The AS/400 only accepts a DB2 for OS/400 user ID and password in uppercase. If a DB2 for OS/400 connection fails due to incorrect authentication, the ODBC driver resends the authentication, forcing the user ID and password into uppercase.

When connecting to DB2 on IBM mainframes, user names and passwords can be of mixed case. The mainframe is not case-sensitive. The ODBC driver sends these values in uppercase.

DB2 Universal Database (UDB) for Windows Server 2003 or for Windows 2000 is case-sensitive. The user ID is stored in uppercase. The password is stored in mixed case and users must enter the password in the correct case. The ODBC driver sends the password exactly in the case entered by the user. The user ID should contain only the user name, not a combination of the Windows NT® domain name and user name.

It is possible to connect using a specific user name and password defined in DB2 on the host system or use the Single Sign-On feature (often referred to as integrated Windows security). If a specific DB2 user name and password are to be used, this information may need to be saved to a data source name (DSN) file. The user name and password are saved in plain text in the DSN file or to registry keys if a System or User DSN is selected. For security reasons when using File DSNs, it is imperative that the DSN file be protected with an access control list (ACL) that restricts access to only authorized users. System and User DSNs are preferred for security reasons if the locations where these ODBC DSNs are stored in the registry have appropriate security protections. Saving the user name and password in the DSN also forces this DSN to be updated whenever the password associated with the user name is changed. So for a variety of reasons, specifying a user name and password is not the preferred authentication option. Using the **Single Sign-On** option is the preferred method for authentication.

Target Database

The **Target Database** tab enables the user to configure required, as well as optional, attributes used to define the target DB2 system.

For the Microsoft ODBC Driver for DB2 in Host Integration Server 2009, the **Target Database** tab has the following fields.

Parameter	Comments
Initial catalog	This parameter is used as the first part of a three-part fully qualified DB2 table name. It is referred to by different names depending on the DB2 platform. In DB2 for OS/390 and DB2 for MVS, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 that you need to connect to on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 Installation Manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, a value can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE.
Package collection	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This can be the same as the default schema. The ODBC Driver for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. The ODBC driver creates packages dynamically in the location that the user points to using the Package Collection parameter. By default, the ODBC Driver for DB2 automatically creates one package in the target collection, if one does not exist, at the time the user issues the first SQL statement. The package is created with GRANT EXECUTE authority to a single <AUTH_ID> only, where AUTH_ID is based on the user ID value configured in the data source. The package is created for use by SQL statements issued under the same isolation level based on the Isolation Level value specified in the connection. Problems can arise in multi-user environments. For example, if a user specifies a Package Collection value that represents a DB2 collection used by multiple users, but this user does not have authority to GRANT execute rights to the packages to other users (the PUBLIC group on the DB2 system, for example), the package is created only for use by this user. This means that other users may be unable to access the required package. The solution is for an administrative user with package administrative rights to create a set of packages for use by all users. (For more information, see Creating Packages for Use with the ODBC Driver for DB2.) The ODBC Driver for DB2 supplied with Host Integration Server 2009 includes the Data Access Tool for use by administrators to create packages. This tool can be run using a privileged User ID to create packages in collections accessed by multiple users. This tool will create a set of packages and grant EXECUTE privilege on these packages to the PUBLIC group representing all users on the DB2 system. The packages (see descriptions under the SQL_ATTR_TXN_ISOLATION connection attribute) created are as follows: AUTOCOMMITTED package (MSNC001 is only applicable on DB2/400) READ UNCOMMITTED package (MSUR001) READ COMMITTED package (MSCS001) REPEATABLE READ package (MSRS001) SERIALIZABLE package (MSRR001) Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400. Once created, the packages are listed in the DB2 (mainframe) SYSIBM.SYSPACKAGE, the DB2 for OS/400 QSYS2.SYSPACKAGE, and the DB2 Universal Database (UDB) SYSIBM.SYSPACKAGE catalog tables. Note that when upgrading from SNA Server 4.0, any existing SNA Server 4.0 packages must be re-created using the Host Integration Server Data Access Tool to make them compatible with Host Integration Server 2009. The package names changed from SNA Server 4.0.

Default Schema	The name of the collection where the ODBC Driver for DB2 looks for catalog information. Default schema is the SCHEMA name for the target collection of tables and views. The ODBC driver uses the Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (for example, ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or owner). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, the ODBC driver uses the USER_ID provided at log on. For DB2/400, the driver uses QSYS2 if no collection is found matching the USER_ID value. This default is inappropriate in many cases so it is essential that the Default Schema value in the data source be defined.
DBMS Platform	The target DB2 platform property value is used to optimize performance of the ODBC driver when executing operations such as data conversion. The default value is DB2/MVS.
Default Qualifier	The name of the schema (collection/owner) with which to fully qualify unqualified object names. This attribute enables the user to access database objects without fully qualifying the objects using a collection (schema) qualifier. The ODBC driver sends this value to DB2 using a **SET CURRENT SQLID** statement, instructing the DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statements. If you do not set a value for the default qualifier, no **SET** statement is issued by the ODBC driver. This ODBC connection attribute is only valid when connecting to DB2 for MVS (OS/390, z/OS).
Alternate TP Name	The Alternate Transaction Program (TP) Name property represents the default transaction program name for the DB2 DRDA application server (AS), which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server 2009 uses the Alternate TP Name in the offline demo configuration (DRDADEMO.UDL). In that case, the Alternative TP Name is set to 0X07F9F9F9.
Distributed transactions	When this option is checked, two-phase commit (distributed unit of work) is enabled. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the Host Integration Server 2009 Resync Service.
Process binary as character	When this option is checked, it indicates that binary data fields should be processed as characters. This option treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. For more information, see the Locale tab.

Locale

The **Locale** tab enables the user to configure the parameters used for character conversion between the client and the DB2 server.

For the Microsoft ODBC Driver for DB2 in Host Integration Server 2009, the **Locale** tab has the following fields.

Parameter	**Comments**
Host CCSID	The coded character set identifier (CCSID) matching the DB2 data as represented on the remote computer. This property is required when processing binary data as character data. Unless the Process Binary as Character value is set, character data is converted based on the DB2 column CCSID and configured ANSI code page. This parameter defaults to U.S./Canada (37).
PC code page	This parameter indicates the personal computer code page to use. It is required when processing binary data as character data. Unless the Process Binary as Character value is set, character data is converted based on the default ANSI code page configured in Windows. The default value for this property is Latin 1 (1252).

Configuration Property Mappings Between the ODBC Driver for DB2 and the OLE DB Provider for DB2

This table compares the configuration parameters used by the ODBC Driver for DB2 and the OLE DB Provider for DB2.

Microsoft ODBC Driver for DB2	Microsoft OLE DB Provider for DB2
General	**General**
Data Source Name	Data Source
Data Source Description	None
Connection	**Connection**
Connection	Network Transport Library
APPC Connection	SNA
TCP/IP Connection	TCPIP
APPC local LU Alias	APPC Local LU Alias
APPC remote LU Alias	APPC Remote LU Alias
APPC mode name	APPC Mode Name
IP address	Network Address
Network port	Network Port
Security	**Security**
Use this username	None
User Name	User ID
Use Single Sign-on	Integrated Security
Affiliate Application	Affiliate Application
Target Database	**Target Database**
Initial catalog	Initial Catalog
Package collection	Package Collection
Default schema	Default Schema
Default qualifier	Default qualifier
DBMS Platform	DBMS Platform
Alternate TP name	Alternate TP Name
Distributed transactions	Distributed transactions
Process binary as character	Process binary as character
Locale	**Locale**
Host CCSID	Host CCSID
PC code page	PC Code Page

Extended	**Extended**
Client Application Name	None
Connection Pooling	None

ODBC Connection String Attributes

The ODBC **SQLBrowseConnect** and **SQLDriverConnect** functions allow passing in a connection string containing a series of attribute/value pairs to the ODBC Driver Manager to establish a connection with a data source. An example of a connection string is as follows:

```
"DSN=MYDATA;NTL=SNA;LLU=Local;RMU=Remote;RDB=BigData;PC=QSYS2;
DS=QSYS2;RO=false;UID=myname;PWD=Secret"
```

Some ODBC attributes are required as part of the connection string when used with the ODBC Driver for DB2.

The following tables compare the configuration parameters used by the ODBC Driver for DB2 and the ODBC attribute keywords that are supported by the OLE DB Driver for DB2 as part of the passed-in connection string.

For the Microsoft ODBC Driver for DB2 in Host Integration Server 2009, these attribute keywords compare as follows.

Microsoft ODBC Driver for DB2	**ODBC attribute keyword**	**Comments**
General	**General**	**General**
Data Source Name	DSN	Required parameter.
Data Source Description	DESC	None.
Connection	**Connection**	**Connection**
Connection	NTL	Required parameter.
APPC connection	NTL=SNA	None.
TCP/IP connection	NTL=TCPIP	None.
APPC local LU alias	LLU	Applicable only if SNA (an APPC connection) is used for the network transport library (NTL=SNA).
APPC remote LU alias	RLU	Applicable only if SNA (an APPC connection) is used for the network transport library (NTL=SNA).
APPC mode name	MN	Applicable only if SNA (an APPC connection) is used for the network transport library (NTL=SNA).
APPC Security Type	AST	Applicable only if SNA (an APPC connection) is used for the network transport library (NTL=SNA). Valid values are PROGRAM (the default), or SAME.
IP address	NA	Applicable only if TCPIP (a TCP/IP connection) is used for the network transport library (NTL=TCPIP).
Network port	NP	Applicable only if TCPIP (a TCP/IP connection) is used for the network transport library (NTL=TCPIP).
Security	**Security**	**Security**
Use this username	None	None.
User Name	UID	None.

	PWD	The Password parameter is not on the Security tab and is not configurable from the ODBC Administrat or tool used to configure ODBC data sources. This parameter can only be preset using the ODBC conne ction string. Most applications prompt the user for this parameter.
Use Single S ign-On	None	Not applicable.
Affiliate app lication	AAP	Applicable only if using Single Sign-On.
Target Dat abase	**Target D atabase**	**Target Database**
Initial Catalo g	RDB	Required parameter.
Package Col lection	PC	Required parameter.
Default Sch ema	DS	Required parameter.
Default Qua lifier	DQ	None.
DBMS Platf orm	DP	Can be one of the following values: DB2/AS400, DB2/MVS (the default), DB2/6000, or DB2/NT.
Alternate TP Name	TPN	None.
Distributed t ransactions	RUW	None.
Process bin ary as chara cter	BAC	None.
Locale	**Locale**	**Locale**
Host CCSID	CCSID	Required parameter.
PC code pag e	CP	Required parameter.

Creating Packages for Use with the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2, which is implemented as an IBM Distributed Relational Database Architecture (DRDA) application requester, uses packages to issue SQL statements and call DB2 stored procedures. There is a configuration parameter that the ODBC Driver for DB2 uses to identify a location in which to create and store DB2 packages. The ODBC Driver for DB2 creates packages dynamically in the location to which the user points using the *Package Collection* parameter. This location may be configured using the **Target Database** tab from the Microsoft ODBC Data Source Administrator tool or can be passed as part of the ODBC connection string as an attribute keyword and argument. The attribute keyword for *Package Collection* is PC.

There are two package creation options:

- The ODBC Driver for DB2 autocreates one package for the currently used isolation level at run time if no package already exists. This auto-create process may fail if the user account does not have authority to create packages.

- An administrator or user can manually create all four packages (five packages on DB2/400) for use with all isolation levels and for use by all users (the PUBLIC group on DB2 representing all users) or a specific set of users. The ODBC Driver for DB2 includes a utility program for use by users with appropriate administrative privilege that will create these packages and grant access to the PUBLIC group for this purpose.

However, some users may not have the security level when manually creating packages to GRANT authority to the packages to other users (grant authority to the DB2 PUBLIC group representing all users, for example). This can be a problem if two or more users with different user IDs try to access a single collection of packages. The first user that created the packages will have access to the packages, but the second user likely will not. The Host Integration Server 2009 CD includes a program for use by an administrator or a user with appropriate privileges to create packages. This tool can be run using a privileged user ID to create packages in collections accessed by multiple users. The Data Access Tool can be used to create packages for use with DB2.

This tool creates a set of packages and grants EXECUTE privileges on these packages to the PUBLIC group. The PUBLIC group on DB2 systems is a default group that represents all DB2 users. The following packages are created:

- AUTOCOMMITTED package (MSNC001), only applicable on DB2/400

- READ UNCOMMITTED package (MSUR001)

- READ COMMITTED package (MSCS001)

- REPEATABLE READ package (MSRS001)

- SERIALIZABLE package (MSRR001)

Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400.

The descriptive process name used by the Data Access Tool corresponds with the isolation levels defined in the ANSI SQL standard. The following table indicates how these packages correspond with the terms used by IBM for isolation levels in DB2 documentation.

Package description	**Package name**	**IBM documentation**
AUTOCOMMITTED (Note that this applies only to DB2/400 and does not correspond with an ANSI SQL isolation level)	MSNC001	COMMIT(*NONE) (NC). This isolation level is used in DB2/400 autocommit mode only and has no corresponding isolation level on other DB2 platforms or in ANSI SQL.

READ UNCOMMITTED	MSUR001	UNCOMMITTED READ (UR). This isolation level corresponds with ANSI SQL READ UNCOMMITTED.
READ COMMITTED	MSCS001	CURSOR STABILITY (CS). This isolation level corresponds with ANSI SQL READ COMMITTED.
REPEATABLE READ	MSRS001	READ STABILITY (RS). This isolation level corresponds with ANSI SQL REPEATABLE READ.
SERIALIZABLE	MSRR001	REPEATABLE READ (RR). This isolation level corresponds with ANSI SQL SERIALIZABLE.

Note that when upgrading from SNA Server 4.0, any existing SNA Server 4.0 packages must be re-created using the Host Integration Server Data Access Tool to make them compatible with Host Integration Server 2009. The package names used by the ODBC Driver for DB2 on SNA Server 4.0 are not compatible with the ODBC Driver for DB2 included with Host Integration Server. On SNA Server 4.0, these packages used different names as follows:

```
AUTOCOMMITTED package (SNANC001) only applicable on DB2/400
READ UNCOMMITTED package (SNACH001)
READ COMMITTED package, (SNACS001)
REPEATABLE READ package, (SNARR001)
SERIALIZABLE package (SNAAL001)
```

These isolation levels are described in detail in Support for Isolation Levels Using the ODBC Driver for DB2. These isolation levels are also described in IsolationLevel Property (ADO). Note that the AUTOCOMMITTED package (MSNC001) is only created on DB2 for OS/400.

Note that the Data Access Tool tool creates this set of packages and grants EXECUTE privileges to PUBLIC. There may be cases for security reasons where EXECUTE privileges to this set of packages should be restricted to a certain group of users or specific users. In these cases, execution privileges on these created packages will need to be modified on the host system.

The Data Access Tool creates all of these packages inside the collection that is specified in the **Package Collection** property in the datalink file, or in the connection string. If the user does not have the appropriate authority to create packages in the specified collection, or if the specified collection does not exist, the ODBC Driver for DB2 returns an error.

In the case of DB2 on MVS or OS/390, the normal error text returned if the user does not have the appropriate authority is as follows:

```
A SQL error has occurred. Please consult the documentation for your specific DB2 version fo
r a description of the associated Native Error and SQL State. SQLSTATE: 51002, SQLCODE: -56
7.
```

In the case of DB2/400, the normal error text returned if the user does not the appropriate authority is as follows:

```
A SQL error has occurred. Please consult the documentation for your specific DB2 version fo
r a description of the associated Native Error and SQL State. SQLSTATE: 51002, SQLCODE: -80
5.
```

In the case of DB2/400, the normal error returned if the collection does not exist is as follows:

```
Failed to create AUTOCOMMITTED (NC) package.  RETCODE=-99.
SQL Error: Code=-204, State=42704, Error Text= A SQL error has occurred.  Please consult th
```

```
e documentation for your specific DB2 version for a description of the associated Native Er
ror and SQL State. SQLSTATE: 42704, SQLCODE: -204
```

There are two authorities required to execute the create package process on OS/390 or MVS using the Data Access Tool:

```
GRANT BINDADD TO <authorization ID>
GRANT CREATE IN COLLECTION <collection ID> TO <authorization ID>
```

The authorization ID is the user who needs the permission to create the packages. The collection ID is the name of the collection, which the user specifies in the datalink file for the **Package Collection** property. This collection should be a valid collection within the DB2.

If an administrator executes the preceding statements on behalf of a nonprivileged user, this nonprivileged user can then run the Data Access Tool. Once run, the process creates four sets of packages (one for each of the four isolation levels supported on DB2 for MVS or OS/390) for use by all (PUBLIC) users of the Microsoft data access features.

The following example illustrates this process on DB2 for MVS or DB2 for OS/390:

Grant rights to run the Data Access Tool to authorization ID WNW999:

```
GRANT BINDADD TO WNW999
GRANT CREATE IN COLLECTION MSPKG TO WNW999
```

Run the Data Access tool using authorization ID WNW999 .

To execute the Data Access Tool on DB2/400, a user ID must have one of the following authorities:

- *CHANGE authority on the DB2 collection
- *ALL authority on the DB2 collection

If the user only has *USE authority or if the user has *EXCLUDE authority, the Create Package process will fail.

There are several steps required to change user authority on a DB2/400 collection (AS/400 library). From interactive SQL (STRSQL command) while logged in as user with administrative privileges, create a new collection. This command can also be issued using ADO, OLE DB, and ODBC. However, most administrators typically create collections from the AS/400 console because the administrator must be logged in at the console to issue the Command Language (CL) command with which to change the user authority on the collection:

```
CREATE COLLECTION <collection ID>
```

From the AS/400 command console, issue the CL WRKOBJ command with the <collection ID> as a parameter:

```
WRKOBJ <collection ID>
```

The collection ID is the name of the collection, which the user specifies in the datalink file for the Package Collection property. This collection should be a valid collection within DB2. The Work with objects window appears. Place the cursor on the *PUBLIC Object Authority line and change the authority from *USE to *ALL.

If an administrator executes the preceding statements on behalf of a nonprivileged user, this nonprivileged user can run the Data Access Tool. Once run, the process creates five sets of packages (one for each of the five isolation levels supported on DB2/400) for use by all (PUBLIC) users of the Microsoft data access features. On DB2/400, five packages are created including the AUTOCOMMITTED packages.

The following example illustrates this process on DB2/400:

Grant rights to run the Data Access Tool to authorization ID WNW999:

```
CREATE COLLECTION MSPKG
WRKOBJ MSPKG
```

Run the Data Access tool.

When using the create package tool, if the package collection specified does not exist, DB2 returns SQLCODE -805.

When using autocreate packages, if a package collection is not specified or the package collection does not exist, during the autocreate package process, the consumer application receives SQLSTATE HY000 and SQLCODE -385. The SQLSTATE HY000 is defined as a driver-specific error. The -385 Error Return Code is not a SQLCODE but rather a DDM DRDA AR (DB2 client) return code. This error code is defined as DDM_VALNSPRM with the following associated text string:

```
"The parameter value is not supported by the target system."
```

Note that when upgrading from SNA Server 4.0, any existing SNA Server 4.0 packages must be re-created using the Host Integration Server Data Access Tool to make them compatible with Host Integration Server 2009.

ActiveX Controls Programmer's Guide

This section provides information about how to integrate your applications using the Data Queue and Host File Transfer ActiveX® controls.

For API references and other technical information about ActiveX controls, see the ActiveX Controls Programmer's Reference section in the SDK.

For sample code using ActiveX controls, see the Data Integration Samples section in the SDK.

This section contains:

- Host File Transfer ActiveX Control Programmer's Guide
- Data Queue ActiveX Control Programmer's Guide

Host File Transfer ActiveX Control Programmer's Guide

The Microsoft Host File Transfer ActiveX control provides the ability to transfer files between a local computer and an MVS, OS/390, AS/400, or AS/36 host system. Host Integration Server 2009 provides this service via a single ActiveX control that depends on other core Host Integration Server DLLs. This extends the ability for a client application to perform file transfer operations from a large number of client development environments.

The Microsoft Host File Transfer ActiveX control uses the record-level input/output (RLIO) protocol of IBM's Distributed Data Management (DDM) architecture to transfer files. The Host File Transfer ActiveX control is implemented as a Distributed Data Management (DDM) source requester, which communicates through APPC LU 6.2 or TCP/IP to a DDM target server.

DDM is a set of rules for distributing or extending data management from one computer to another, such as from a mainframe to an AS/400 computer or from one of these host computers to a server computer. By combining the Microsoft File Transfer ActiveX control and DDM architectures, Microsoft enables organizations to preserve their investments in existing data management infrastructure, while extending universal file transfer to all enterprise-wide data sources.

The information in this section is required to develop applications with Host Integration Server that use ActiveX or COM objects to transfer files from local machines to hosts in a Systems Network Architecture (SNA) environment or over TCP/IP using RLIO and DDM.

In This Section

Platforms Supported by the Host File Transfer ActiveX Control

On the mainframe platform, IBM offers a target DDM server implementation in IBM Distributed File Manager (DFM), a component of IBM Data Facility Storage Management Subsystem (DFSMS).

On the mainframe platform, the Host File Transfer ActiveX control supports the following data set types:

Sequential Access Method (SAM) data sets

- Basic Sequential Access Method data sets (BSAM)
- Queued Sequential Access Method data sets (QSAM)

Basic Partitioned Access Method (PDS) data sets

- Partitioned Data Set Extended members (PDSE)
- Partitioned Data Set members (PDS)

Virtual Storage Access Method (VSAM) data sets

- Entry-Sequenced Data Sets (ESDS)
- Key-Sequenced Data Sets (KSDS)
- Fixed-Length Relative Record Data Sets (RRDS)
- Variable-Length Relative Record Data Sets (VRRDS)
- Relative Record Data Set (RRDS)
- VSAM Alternate Indexes for ESDS and KSDS data sets

The preceding data set types are supported by IBM DFM/MVS. The following data set types are not supported by DFM/MVS and cannot be accessed using the Host File Transfer ActiveX control.

- VSAM Linear Data Sets (LDS)
- Generation Data Groups (GDG)
- Generation Data Sets (GDS)
- Basic Direct Access Method data sets (BDAM)
- Indexed Sequential Access Method data sets (ISAM)
- Sequential Data Striping data sets
- OpenEdition MVS Hierarchical File System (HFS) files
- Tape Media

All mainframe data sets accessible through IBM Distributed File Manager must be cataloged in an Intersystem communications function (ICF) catalog and reside on direct access storage devices (DASD).

Configuring Data Descriptions for Host File Transfer

In order to use the Microsoft® Host File Transfer ActiveX® control to transfer files, a user or client application must describe the data format of the host file to transfer. A host data description is normally configured using the Data Access Tool.

The Data Access Tool contains one high-level object for configuring Host Column Description files:

- **Host Column Descriptions**—Stored in Host Column Description (HCD) files that contain the information required to convert host data types to PC computer data types.

When creating a Data Description for use with the Host File Transfer ActiveX control, the **Use Table for File Transfer** check box must be selected and values must be entered for the following additional parameters:

Parameter	Comment
FieldDelimiter	This value represents the delimiter used to mark the end of one field and the beginning of another within a single record. The position of the field elements within a record is assumed to be absolute as configured and variable-length fields are not supported. This element is used in order to inform the conversion routine to remove this character if it is found at the position within the record as indicated by the Data Description. This parameter is required and has no default value. The comma character "," or tab character "\t" is commonly used with desktop applications as a field separator.
RecordDelimiter	This value represents the character or characters that appear at the ending of a record. This element is used in order to inform the conversion routine to remove this character if it is found in the last position of the record as indicated by the Data Description. This parameter is required and has no default value. The end-of-line character sequence is commonly used as a record delimiter. The carriage return and linefeed character sequence "\n\r" is commonly used with desktop applications. The newline character "\n" is standard for use on UNIX systems and with some desktop applications.
TextQualifier	This value is used in order to allow elements that may contain the **FieldDelimiter** character to be properly parsed by the parsing engine. If this element is enabled on a particular field, then the inclusion of this character at the beginning of the field is an indication to the parsing engine that all characters up to the next instance of the **TextQualifier** should be treated as part of the current field being processed. This parameter has no default value. The single quote or double quote character is sometimes used as a text qualifier to protect a comma character included in a field from being misinterpreted as a field delimiter.

Registry Settings Used By Host File Transfer

The Microsoft® Host File Transfer ActiveX® control uses a number of registry settings for configuration and proper operation. The configuration registry settings are located under the HKEY_LOCAL_MACHINE\Software\Microsoft\SNA Server\CurrentVersion\Setup key. These registry settings include the following subkey:

Subkey	Comment
RootDir	Stores the path to root directory where the Host Integration Server was installed. The system directory below this root directory is the location where the Host File Transfer ActiveX control DLL and other support DLLs are installed.

Object Support Using Host File Transfer

The Microsoft® Host File Transfer ActiveX® control supports a number of standard COM interfaces as well some custom objects and interfaces.

This section contains:

- COM Interface Support Using Host File Transfer
- IEIGFileTransferCtl Object
- IEIGFileTransferCtlEvents Notification

COM Interface Support Using Host File Transfer

The Microsoft Host File Transfer ActiveX control supports a number of standard COM interfaces as well as a single custom interface, **IEIGFileTransferCtl**. The ActiveX control object has the ability to register and de-register itself via standard control mechanisms. Support for a number of standard COM interfaces makes it easy to develop applications using the Host File Transfer ActiveX control with Visual Basic and Visual C++ as well as from Internet Explorer and Access. Supporting a variety of standard COM interfaces also provides different ways for a client to save information.

The following table summarizes the standard COM interfaces supported by the Host File Transfer ActiveX control.

COM interface	Comments
ICategorizeProperties	This interface divides up the properties into an intelligent presentation to the client.
IConnectionPointContainer	None.
IDispatch	A dual interface deriving from **IDispatch** is exposed to provide support and flexibility to clients. Clients that provide support for automation interfaces will utilize the **IDispatch** interface, while more robust clients may use the custom interface. Using the custom interface provides for the greatest execution speed.
IOleControl	None.
IOleInPlaceActiveObject	None.
IOleInPlaceObject	None.
IOleInPlaceObjectWindowless	None.
IOleObject	None.
IOleWindow	None.
IPerPropertyBrowsing	This interface provides support for client browsing of properties in an intelligent manner. This interface exposes to the client property lists used in the population of a dropdown list. This interface is required for the control to be hosted by Microsoft Access.
IPropertyNotifySink	The interface is implemented by a sink object to receive notifications about property changes from an object that supports **IPropertyNotifySink** as an outgoing interface. The client that needs to receive the notifications in this interface (from a supporting connectable object) creates a sink with this interface and connects it to the connectable object through the connection point mechanism.
IPersist	None.

IPersistPropertyBag	This interface is the preferred method of property persisting for Internet Explorer and Visual Basic. Using this interface, persisted properties are stored as a set of name/VARIANT value pairs.
IPersistStorage	This interface stores persistent properties into a structured storage object.
IPersistStreamInit	This interface is responsible for saving the persisted properties in binary form using a stream interface. This is the method used by the Microsoft Visual C/C++ compiler to persist properties.
IProvideClassInfo	None.
IProvideClassInfo2	None.
IQuickActivate	None.
ISupportErrorInfo	This interface is the preferred method to return error indications to scripting clients. Using this interface, error codes and explanation text are returned to the client. This information may be used in order to provide diagnostic information to the user and in cases of failure.
IViewObject	None.
IViewObject2	None.
IViewObjectEx	None.

IEIGFileTransferCtl Object

The Microsoft Host File Transfer ActiveX control supports a number of standard COM interfaces as well as a single custom interface. The **IEIGFileTransferCtl** object supports a number or properties and methods that provide the ability to transfer files to and from MVS, OS/390, AS/400, or AS/36 hosts. The Host File Transfer ActiveX control also supports a set of events notifying a client application of connection status, file transfer status, and error reporting. These events are handled by the client supporting several callback functions and setting these callbacks using the IConnectionPointContainer.

The following IEIGFileTransferCtl object methods are supported by the Microsoft Host File Transfer ActiveX control:

Method name	Comment
Cancel method	Terminates a file transfer operation that is already in progress.
Connect method	Establishes a connection to the configured host and reports to the user an indication of the success or failure of the action.
Disconnect method	Terminates an existing connection to a host machine.
GetFile method	Copies a file from host storage to local storage. This method requires the two file names as parameters.
PutFile method	Copies a file from local storage to host storage. This method requires the two file names as parameters.

The following **IEIGFileTransferCtl** object properties are supported by the Microsoft Host File Transfer ActiveX control:

Property name	Comment
AppendToEnd property	Sets or returns whether a file transfer should append to the end of a file (eigAnswerYes) if the file exists, or should it overwrite the existing contents replacing the data with the new information (eigAnswerNo). This property defaults to eigAnswerYes (0).
CCSID property	Sets or returns the character code set identifier (CCSID) that must match the data in the file as represented on the remote host computer. This property defaults to U.S./Canada (37).
ConnectionState property	Returns the current state of the connection. The state of a connection can be unspecified, idle, connecting, connected, or disconnecting.
ConnectionType property	Sets or returns the network transport used for this connection. The **ConnectionType** property designates whether the Host File Transfer ActiveX control connects through APPC (SNA LU 6.2) or TCP/IP. The possible values for this parameter are a TCP/IP or an APPC connection using an enumerated value. The default value for this parameter is an APPC (SNA) connection type. If APPC is selected, then values for the **LocalLU**, **ModeName**, and **RemoteLU** properties are required. If TCP/IP is selected, then values for the **NetAddr** and **NetPort** properties are required.
CreateIfNonExisting property	Sets or returns whether a file operation should create a new destination file if one does not already exist (eigAnswerYes). This property defaults to eigAnswerNo (1)
LocalLU property	Sets or returns the Local LU Alias. When LU 6.2 (SNA) is selected for the ConnectionType property, this property must match the name of the local LU alias configured using SNA Manager. This property defaults to the string value of "LOCAL" represented as a BSTR.

ModeName property	Sets or returns the APPC mode. When APPC (LU 6.2 SNA) is selected for the ConnectionType property, this field must bet set to the APPC mode that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This property defaults to the string value of "QPCSUPP" represented as a BSTR.
NetAddr property	Sets or returns the IP address of the host computer. When TCP/IP has been selected for the **ConnectionType** property, this property indicates the IP address of the host. This property can be an IP address or the name representing the host IP address using the Domain Name System (sna.microsoft.com, for example). This property is a string (BSTR) and has no default value.
NetPort property	Sets or returns the TCP/IP port used for communication with the host. When TCP/IP has been selected for the ConnectionType property, this parameter is the TCP/IP port used for communication with the host. The default value for this property is the string (BSTR) "446" representing TCP/IP port 446.
OverwriteHostFile property	Sets or returns whether a file operation request to copy a file that will write over an existing file will fail. When this property is set to eigAnswerNo, a request to write a file over an existing file will fail. This property defaults to eigAnswerNo (1)
Password property	Sets or returns the password used for authentication. A valid user name and password are normally required to access files on a host computer. The password is case sensitive and is normally displayed as asterisks in a dialog box for security purposes. This property is a string (BSTR) and has no default value.
PCCodePage property	Sets or returns the PC codepage The **PCCodePage** property indicates the codepage to be used on the PC for character code conversion. This property defaults to Latin 1 (1252).
RDBName property	Sets or returns the name of the remote database name and the host column description (HCD) file that describes the data types and data conversions used to transfer this file. The HCD file describing the data should be located in the system subdirectory below the root directory where Host Integration Server was installed. Setup defaults to the following location: C:\Program Files\Host Integration Server When TCP/IP is selected for the **ConnectionType** property, the **RDBName** must also match the name of the remote database system.
RemoteLU property	Sets or returns the Remote LU Alias. When APPC (LU 6.2 SNA) is selected for the **ConnectionType** property, this property is the name of the local LU alias configured using SNA Manager. This property is a string (BSTR) has no default value.
UserID property	Sets or returns the user name used for authentication. A valid user name and password are normally required to access files on a host computer. This value is case sensitive. This property is a string (BSTR) and has no default value.

IEIGFileTransferCtlEvents Notification

The Microsoft Host File Transfer ActiveX control also supports a set of events notifying a client application of connection status, file transfer status, and error reporting. These events are handled by the client supporting several callback interfaces and setting these callbacks derived from the standard **IConnectionPointContainer** COM object.

The following IEIGFileTransferCtlEvents notification interface methods are supported by the Microsoft Host File Transfer ActiveX control:

Event notifications	Comment
ConnectionStateChange	This event is fired when the state of a connection has changed. A *ConnectionState* parameter is passed to the client callback function that receives this event method call. This parameter is an eigConnectionStateEnum value representing the new state of the ConnectionState property.
ReportError	This event is used in order to return error conditions that occur during the synchronous processing of methods. This event passes two parameters to the client callback function that receives this event method call. The first parameter is a long value representing an error code. The second parameter is a BSTR string containing a brief text description of the error.
TransferComplete	This event is an indication to the client that the requested transfer operation has completed.
TransferProgress	This event will be fired periodically in order to inform the client of the progress of an unattended file transfer. A *PercentageDone* parameter is passed to the client callback function that receives this event method call. This parameter is a short value representing the percentage complete of the requested operation ranging from 0 to 100. A client application has the option to terminate a file transfer when this event is fired.

Programming Considerations When Using Host File Transfer

The Microsoft® Host File Transfer ActiveX® control exposes a dual interface deriving from **IDispatch**. This provides support and flexibility to clients wishing to use the object. Clients that provide support automation interfaces can use the **IDispatch** interface while more robust clients may use the custom interface. Using the custom interface offers the greatest execution speed.

The single-threading model is supported, allowing only single threads to access the objects safely.

The Host File Transfer ActiveX control does not support uploading Direct Relative Record Data Set (RRDS) files on System/36. An error (381) will occur and the following error message will be received.

```
"Target Not Supported"
```

The Host File Transfer ActiveX control also does not support uploading RRDS files with the CreateIfNonExisting property option set to yes on System/36. An error will occur (381) and the following message will be received.

```
"Target Not Supported"
```

If there is existing data in a file on OS/390, setting the OverwriteHostFile and AppendToEnd properties to "no" should cause an error (58) to occur and the following error message should be received.

```
"File contains existing data. Upload not configured to append data - upload aborted"
```

On OS/390, this error is not triggered for sequential or KSDS files. Instead the data is appended to the existing data in the file for sequential files and duplicate records are not appended to KSDS files.

The **AppendToEnd** property and the **OverwriteHostFile** property are mutually exclusive, so it is not possible to enable (set to yes) one of these properties before the opposing property is disabled (set to no). The **AppendToEnd** property takes precedence over the **OverwriteHostFile** property, since **AppendToEnd** defaults to yes and **OverwriteHostFile** defaults to no. Consequently, the order in which these properties are set will affect the outcome. For example, the following order will result in the properties being set correctly:

```
FileTransfer.AppendToEnd = eigAnswerNo         // correctly set to no
FileTransfer.OverwriteHostFile = eigAnswerYes  // correctly set to yes
```

In contrast, setting the properties in the improper order will cause the properties to be set incorrectly as follows:

```
FileTransfer.OverwriteHostFile = eigAnswerYes   // remains at no
// AppendToEnd defaults to eigAnswerYes, so this change is illegal
FileTransfer.AppendToEnd = eigAnswerNo          // correctly set to no
```

In this second case, the **OverwriteHostFile** property cannot be set to yes (enabled) until the **AppendToEnd** property is set to no (disabled).

Using the Data Descriptions tool, setting the Ascending/Descending option on an OS/390 or MVS/ESA key sequenced file has no effect. Using the Host File Transfer ActiveX control, data is always uploaded and downloaded in the ascending key order.

If the **Cancel** method is executed while uploading a file with the **AppendToEnd** property set to yes, this will result in no change to the host file. However, if the **Cancel** method is executed while uploading a file with the **OverwriteHostFile** property set to yes, this will result in an empty host file. The **Cancel** method implies the transfer has been stopped and all the files are at their original values, but this is not really the case when the **OverwriteHostFile** property is set to yes.

This section contains:

- Code Page Support Using Host File Transfer
- Data Conversion Using Host File Transfer
- Usernames and Passwords Using Host File Transfer

- Troubleshooting the Host File Transfer ActiveX Control

Code Page Support Using Host File Transfer

When using the Host File Transfer ActiveX control, the Host CCSID (character code set identifier) property should be configured to match the data as represented on the remote host computer. The Host CCSID parameter defaults to EBCDIC U.S./Canada (37) when using the Host File Transfer ActiveX control.

This section contains:

- ISO Code Page Support Using Host File Transfer
- DBCS Code Page Support Using Host File Transfer

ISO Code Page Support Using Host File Transfer

Host Integration Server 2009 includes support for some ISO code pages for purposes of ISO-to-UNICODE-to-ANSI, ANSI-to-UNICODE-to-ISO, and ISO-to-UNICODE-to-ISO conversions when using the Host File Transfer ActiveX control. These ISO code pages can be used when accessing host files containing ISO code pages.

Depending on the version of Windows being used, to support ISO-to-UNICODE-to-ANSI (Windows), ANSI-to-UNCODE-to-ISO, and ISO-to-UNICODE-to-ISO code page conversions, you may need to install the appropriate ISO National Language Support (NLS) file for your locale.

On Microsoft Windows 2000, the appropriate ISO NLS file for your locale is installed automatically when you install a localized version of Windows 2000. On Windows Server 2003, the appropriate ISO NLS file for your locale is installed automatically when you install the US-English version.

The following table shows the ISO character code set identifiers (CCSIDs) supported by Host File Transfer ActiveX control in Host Integration Server 2009.

Microsoft display name	Microsoft NLS code page	IBM CCSID	Comments
ISO 8859-1 Latin 1	28591	819	
ISO 8859-2 Central europe	28592	912	
ISO 8859-5 Cyrillic	28595	915	
ISO 8859-6 Arabic	28596	1089	
ISO 8859-7 Greek	28597	813	
ISO 8859-8 Hebrew	28598	916	
ISO 8859-9 Turkish	28599	920	
ISO 6937 Non-Spacing Accent	20269	819	Note that ISO 6937 (CCSID 20269) is not supported by the OLE DB Provider for DB2, but is displayed in the list of configuration options when creating or modifying data sources.
ISO 8859-15 Latin 9 (euro)	20865	923	NLS Code Page 819 with support for the euro.

The Microsoft display name is the name found in the Windows 2000 definitions for these NLS files.

The Microsoft NLS code page column represents the code page number that is registered and associated with an ISO-to-UNICODE NLS resource file. The Microsoft NLS number should be set as the Host CCSID when using the Host File Transfer ActiveX control. When setting the Host CCSID or PC Code Page property, use the Microsoft NLS number for this parameter.

The IBM CCSID column represents the CCSID given to the ISO code page in IBM publications. IBM lists their ISO support in publications by referencing the locale name (Bulgaria for ISO8859-5 and 915, for example) rather than simply using ISO 8859-5 Cyrillic as used by Microsoft. The Host File Transfer ActiveX control does not recognize or display the IBM CCSID values.

The Host File Transfer ActiveX control maps the Microsoft NLS numbers to ISO NLS files which correspond with the appropriate IBM CCSID numbers. The Host File Transfer ActiveX control passes the corresponding IBM CCSID to the host system at run time even though you configure this property using the Microsoft NLS number.

Note

The IBM CCSID 819 is associated with both ISO 8859-1 Latin 1 and ISO 6937 Non-Spacing Accent. It is up to the user to choose the standard ISO 8859-1 Latin 1 code page by selecting NLS code page 28591 or the modified code page ISO 6937 Non-Spacing Accent by selecting NLS code page 20269.

Note

The ISO 6937 Non-Spacing Accent (CCSID 20269) is not currently supported by the Host File Transfer ActiveX control.

IBM CCSID 916 (ISO 8859-8) supports Hebrew "visual sort order". IBM CCSID 920 (ISO 8859-8 derivation) supports Hebrew "logical sort order". Although Microsoft supports the logical sort order with NLS 38598, this NLS file is only distributed with Internet Explorer 5 or Windows 2000.

The Host File Transfer ActiveX control has not been tested using the ISO 8859-8 derivation matching IBM CCSID 920 and does not support this configuration.

These are the only ISO pages currently supported in Host Integration Server 2009. Microsoft supports a number of additional ISO pages. IBM also supports additional ISO pages. However, the code pages listed in the table above are the only cases where the Microsoft NLS pages and IBM CCSIDs match.

DBCS Code Page Support Using Host File Transfer

Support for Double-Byte Character String (DBCS) data is limited using the Host File Transfer ActiveX control. Conversions between DBCS and ANSI code pages are not supported. Conversions between DBCS and ISO code pages are not supported.

Data Conversion Using Host File Transfer

Using the Host File Transfer ActiveX control, host data is converted to default C data types as defined in ODBC and OLE DB and illustrated in the following table:

Host data type (description in HCD file)	Default C data type	Comments
BINARY		A free-form binary data type of specified length. This data type is transferred without being converted.
CHAR	char string[]	A fixed-length string. This data type is converted to a DBTYPE_BSTR for use by Host File Transfer ActiveX control.
DATE	date struct	A 10-byte date string. This data type is converted to a DBTYPE_DATE for use by OLE DB.
DOUBLE	double	An 8-byte double-precision floating-point number. This data type is converted to a DBTYPE_R8 for use by OLE DB.
FLOAT	double	An 8-byte double-precision floating point number. This data type is the same as a DOUBLE. This data type is converted to a DBTYPE_R8 for use by OLE DB.
LONG	int	A 4-byte integer ranging in value from -2,147,463,648 to +2,147,483,647. This data type is converted to a DBTYPE_I4 for use by OLE DB.
LONG VARBINARY	char string[]	A varying-length binary string up to 32,740 bytes in length. This data type is converted to a DBTYPE_STR for use by OLE DB.
LONG VARCHAR	char string[]	A varying-length character string up to 32,740 characters in length. This data type is converted to a DBTYPE_STR for use by OLE DB.
PACKED	unsigned char number[]	A packed decimal number. This data type is converted to a DBTYPE_DECIMAL for use by OLE DB
REAL	float	A 4-byte single-precision floating-point number. This data type is converted to a DBTYPE_R4 for use by OLE DB.
SHORT	short	A SMALLINT (small integer) is a two-byte integer with a precision of 5 digits ranging from -32,768 to +32,767. This data type is converted to a DBTYPE_I2 for use by OLE DB.
SINGLE	float	A 4-byte single-precision floating-point number. This data type is converted to a DBTYPE_R4 for use by OLE DB.
TIME	time struct	An 8-byte time string. This data type is converted to a DBTYPE_TIME for use by OLE DB. When using ActiveX Data Objects to return data from a DB2 TIME data type, ADO returns a DATETIME value.

TIMESTAMP	timestamp struct	A 26-byte string representing the date, time, and microseconds. This data type is converted to a DBTYPE_DBTIMESTAMP for use by OLE DB.
VARBINARY	char string[]	A varying-length binary field. The maximum length of the binary is dependent on the version and the host platform. This data type is transferred without being converted. This data type is converted to a DBTYPE_STR for use by OLE DB.
VARCHAR	char string[]	A varying-length character string. The maximum length of the string is dependent on the version and the host platform. This data type is converted to a DBTYPE_STR for use by OLE DB.
ZONED	unsigned char number[]	A zoned numeric number. This data type is converted to a DBTYPE_NUMERIC for use by OLE DB

Note

The maximum length of fixed-length BINARY, fixed-length CHAR, VARBINARY, and VARCHAR data types is dependent on the version of the host software that is being accessed. For example, the maximum length of the CHAR data type on OS/390 is 254 characters, while the maximum length of this same host data type is 32,765 on OS/400.

Data conversions from a large numeric type to a small numeric type are supported (from DOUBLE to SINGLE and from INT to SMALLINT, for example), however truncation and conversion errors can occur that will not be reported by the Host File Transfer ActiveX control.

Using the Host File Transfer ActiveX control, certain conversions of strings from EBCDIC to ASCII and then back to EBCDIC are asymmetric, and can result in strings that are different from the original. The EBCDIC specification contains ordinals for which there is no defined character. The Host File Transfer ActiveX control translates all such undefined characters to the question mark character ("?"). So when ASCII strings containing these characters are converted back to EBCDIC, these undefined characters will be replaced with question marks. To protect EBCDIC strings containing undefined characters, these fields should be tagged as binary strings and mapped by the application.

The ANSI to EBCDIC character conversions affected include the following:

Character value (decimal)	Character value (hexadecimal)	ANSI code page 1252	EBCDIC character after conversion to CCSID 37
128	0x80	Not used	?
130	0x82	Single low quote	?
131	0x83	Latin F with hook	?
132	0x84	Double low quote	?
133	0x85	Ellipsis	?
134	0x86	Dagger	?
135	0x87	Double dagger	?
136	0x88	Per mile	?
137	0x89	S with caron	?
138	0x8A	Left angle	?
139	0x8B	Ligature OE	?
140	0x8C	Not used	?

142	0x8E	Not used	?
145-156	0x91-0x9C		?
158-159	0x9E-0x9F		?

Usernames and Passwords Using Host File Transfer

When connecting to host systems, most users must be authenticated by the remote system by passing a valid user ID and password.

The AS/400 computer is case-sensitive with regard to user ID and password. The AS/400 only accepts a user ID and password in uppercase. The Microsoft Host File Transfer ActiveX control forces the User ID and Password into uppercase when it knows that it is connecting to an AS/400 system.

The mainframe is not case-sensitive. This means that on mainframe computers, one can enter the user ID and password in any case.

Troubleshooting the Host File Transfer ActiveX Control

The Microsoft Host File Transfer ActiveX control supplied with Host Integration Server 2009 has the ability to trace DRDA data flows when used over TCP/IP.

This tracing capability is accessible from the SNADDM Service tracing inside the Trace tool. This facility will show the same data as an APPC trace but without the control indicators (For example, What_Received). Socket errors are traced and the error codes can be looked up in Winsock2.h supplied with the Platform SDK.

The Host File Transfer ActiveX control can return the following types of errors:

- Errors from the remote hosts
- Microsoft Host File Transfer-specific errors
- Errors from the underlying DDM application requester network client

Data Queue ActiveX Control Programmer's Guide

A data queue is an AS/400 system object that is used for inter-process communications between multiple programs or jobs. Data queues allow multiple programs to send and receive shared messages via a central repository without first writing the message data to a physical database file. Typically, when a data record is read from the queue, the record is erased from the queue. The advantage of using data queues to share data in comparison with using database files is that data queues require much less file I/O and therefore improve overall system performance.

The Microsoft Data Queue ActiveX control provides the ability to access AS/400 data queues. Microsoft Host Integration Server 2009 provides this service via a single ActiveX control that depends on other core Host Integration Server DLLs. Developers can move part or all of their AS/400 applications from an AS/400 computer to a PC platform, while retaining access in the program running on the PC to a remote data queue on the AS/400.

The Data Queue ActiveX control is implemented as a Distributed Data Management (DDM) Application Requester. The Data Queue ActiveX control uses the Data Queue interfaces in the DDM Level 4 architecture, which are extensions to the record-level input/output (RLIO) protocol of IBM's Distributed Data Management architecture.

DDM is a set of rules for distributing or extending data management from one computer to another, such as from a mainframe to an AS/400 computer or from one of these host computers to a server computer. By combining the Microsoft Data Queue ActiveX control and DDM architectures, Microsoft enables organizations to preserve their investments in existing data management infrastructure, while extending universal file and data transfer to all enterprise-wide data sources.

The information in this section is required to develop applications with Host Integration Server that use ActiveX or COM objects to transfer data from local machines to AS/400 Data Queues in a Systems Network Architecture (SNA) environment.

In This Section

Advantages of Data Queues

Data queues provide a fast means of inter-process communication, requiring low system overhead and minimal setup. AS/400 Data Queues are designed to provide a flexible, highly efficient, yet temporary means of inter-process communication. Data queues are familiar to most AS/400 programmers as a simple method of passing information to another program.

Data queues provide considerable flexibility to the application programmer. The data queues interfaces require no communications programming and can be used either for connected or disconnected communication. AS/400 and PC applications can be developed using any supported language, yet still communicate with each other. PC programs can communicate with AS/400 programs through a common AS/400 data queue. The use of data queues requires little knowledge of communication and no knowledge of APPC if the programmer utilizes the Microsoft® Data Queue ActiveX® control. The data queue messages are merely described at the record level, allowing the application programmer to define the field-level structure as required.

By default, when one program reads an entry in the queue, the entry is then deleted. Pointers to the queue entries are then updated to reflect the change in the record stack. A data queue can exist with no entries, a single entry, or multiple entries. Multiple concurrent jobs and programs can access data queues.

When receiving data, the requesting application can set a time-out value to wait for data to arrive in the queue. Waits can be applied based on entry of the data record or for a time period (zero seconds to many days in length). A program that reads from a queue need not be running when the queue is created or when records are inserted. A single data queue can support many separate interactive jobs. At regular intervals or at the end of the day, records in the data queue can be persisted to a file by a single automated batch process.

See Also
Other Resources
Data Queue ActiveX Control Programmer's Guide

Object Support Using Data Queues

The Microsoft® Data Queue ActiveX® control supports a number of standard COM interfaces as well some custom objects and interfaces.

This section contains:

- COM Interface Support Using Data Queues
- IEIGDataQueueCtl Object
- IEIGDataQueue Object
- IEIGDataQueueItem Object
- IEIGDataQueueCtlEvents Notifications
- IEIGDataQueueEvents Notifications

COM Interface Support Using Data Queues

The Microsoft Data Queue ActiveX control supports a number of standard COM interfaces as well as several custom interfaces, **IEIGDataQueueCtl**, **IEIGDataQueue**, and **IEIGDataQueueItem**. The ActiveX control object has the ability to register and de-register itself via standard control mechanisms. Support for a number of standard COM interfaces makes it easy to develop applications using the Data Queue ActiveX control with Visual Basicand Visual C++ as well as from Internet Explorer and Access. Supporting a variety of standard COM interfaces also provides different ways for a client to save information.

The following table summarizes the standard COM interfaces supported by the Data Queue ActiveX control.

COM interface	Comments
ICategorizeProperties	This interface divides up the properties into an intelligent presentation to the client.
IConnectionPointContainer	
IDispatch	A dual interface deriving from **IDispatch** is exposed to provide support and flexibility to clients. Clients that provide support for automation interfaces will utilize the **IDispatch** interface, while more robust clients may use the custom interface. Using the custom interface provides for the greatest execution speed.
IPerPropertyBrowsing	This interface provides support for client browsing of properties in an intelligent manner. This interface exposes to the client property lists used in the population of a drop-down list. This interface is required for the control to be hosted by Microsoft Access.
IPropertyNotifySink	The interface is implemented by a sink object to receive notifications about property changes from an object that supports **IPropertyNotifySink** as an outgoing interface. The client that needs to receive the notifications in this interface (from a supporting connectable object) creates a sink with this interface and connects it to the connectable object through the connection point mechanism.
IPersist	
IPersistPropertyBag	This interface is the preferred method of property persisting for Internet Explorer and Visual Basic. Using this interface persisted properties are stored as a set of name/VARIANT value pairs.
IPersistStorage	This interface stores persistent properties into a structured storage object.
IPersistStreamInit	This interface is responsible for saving the persisted properties in binary form using a stream interface. This is the method used by the Microsoft Visual C/C++ compiler to persist properties.
ISupportErrorInfo	This interface is the preferred method to return error indications to scripting clients. Using this interface, error codes and explanation text are returned to the client. This information may be used in order to provide diagnostic information to the user and in cases of failure.

See Also

Other Resources

Object Support Using Data Queues

IEIGDataQueueCtl Object

The **IEIGDataQueueCtl** object supports a number of properties and methods that provide the ability to connect with a host and communicate with OS/400 Data Queues. The **IEIGDataQueueCtl** also supports a set of events notifying a client application of connection status and error reporting. These events are handled by the client supporting several callback functions and setting these callbacks using the **IConnectionPointContainer**.

The following **IEIGDataQueueCtl** object methods are supported by the Microsoft Data Queue ActiveX control.

Method name	Comment
Connect	Establishes a connection to the configured host and reports to the user an indication of the success or failure of the action.
CreateQueueContainer	Creates an instance of an IEIGDataQueue container object and optionally initializes the QueueName property. The created queue object is assumed to be associated with the connection object that created it for the life of the connection or the life of the queue object.
Disconnect	Terminates an existing connection to a host machine.

The following **IEIGDataQueueCtl** object properties are supported by the Microsoft Data Queue ActiveX control.

Property name	Comment
CCSID	Sets or returns the character code set identifier (CCSID) that must match the data in the AS/400 data queue as represented on the remote host computer. This property defaults to U.S./Canada (37).
ConnectionState	Returns the current state of the connection. The state of a connection can be unspecified, idle, connecting, connected, or disconnecting.
ConnectionType	Sets or returns the network transport used for this connection. The default value for this parameter is SNA. Note that TCP/IP is not supported. If APPC is selected, then values for the **LocalLU**, **ModeName**, and **RemoteLU** properties are required.
LocalLU	Sets or returns the Local LU Alias. When APPC (SNA LU 6.2) is selected for the ConnectionType property, this property must match the name of the local LU alias configured using SNA Manager. This property defaults to the string value of "LOCAL" represented as a BSTR.
ModeName	Sets or returns the APPC mode. When APPC (LU 6.2 SNA) is selected for the **ConnectionType** property, this field must be set to the APPC mode that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This property defaults to the string value of "QPCSUPP" represented as a BSTR.
Password	Sets or returns the password used for authentication. A valid user name and password are normally required to access data on a host computer. The password is case sensitive and is normally displayed as asterisks in a dialog box for security purposes. This property is a string (BSTR) and has no default value.
PCCodePage	Sets or returns the PC code page The PC Code Page property indicates the code page to be used on the PC for character code conversion. This property defaults to Latin 1 (1252).

RemoteLU	Sets or returns the Remote LU Alias. When APPC (LU 6.2 SNA) is selected for the **ConnectionType** property, this property is the name of the remote LU alias configured using SNA Manager. This property is a string (BSTR) and has no default value.
UserID	Sets or returns the user name used for authentication. A valid user name and password are normally required to access data on a host computer. This value is case sensitive. This property is a string (BSTR) and has no default value.

See Also

Other Resources

Object Support Using Data Queues

IEIGDataQueue Object

The **IEIGDataQueue** object represents a logical queue and supports a number of properties and methods that provide the ability to communicate with a specific data queue. The QueueName property is the name of the physical queue. All methods and events are related to the queue that is represented by the individual instance of the object. The Data Queue ActiveX control also supports a set of events notifying a client application of connection status, data transfer status, and error reporting. These events are handled by the client supporting several callback functions and setting these callbacks using the **IConnectionPointContainer**.

The following **IEIGDataQueue** object methods are supported by the Microsoft Data Queue ActiveX control.

Method name	**Comment**
AddQueueItem	Adds a record to the current queue.
Cancel	Terminates a request to receive information from the queue that is already in progress.
CancelQueue	Indicates that an application no longer wants to be notified of an incoming queue data item. This can be used to stop pending notifications that were queued as a result of calling **GetQueueItem**.
ClearAll	Removes all items from the queue.
CreateQueue	Creates a data queue.
DeleteQueue	Clears all messages from the queue and then deletes the queue.
GetQueueItem	Retrieves an item from the queue.
QueryAttribute	Requests information on one of the queue's attributes
SetAttribute	Changes the attributes associated with a data queue.
StopQueue	Stops the queue from responding to client requests.

The following **IEIGDataQueue** object property is supported by the Microsoft Data Queue ActiveX control.

Property name	**Comment**
QueueName	This is the name of the data queue this object is associated with.

See Also

Other Resources

Object Support Using Data Queues

IEIGDataQueueItem Object

The **IEIGDataQueueItem** object represents a specific queue item and supports a number of properties and methods.

The following **IEIGDataQueueItem** object method is supported by the Microsoft Data Queue ActiveX control.

Method name	Comment
Reset	Resets the queue item properties to default values.

The following **IEIGDataQueueItem** object properties are supported by the Microsoft Data Queue ActiveX control.

Property name	Comment
ExtUser	The external job user name.
ExtJobName	The external job name.
ExtJobNumber	The external job number.
InactiveRec	Indicates an Inactive record.
Keyval	The key value.
Message	The queue message.
QItemType	The type of queue item this represents.
Record	The entire queue data.
RecordAttribute	The list of record attributes.
RecCount	The record count.
RecNumber	The record number.
ReplyRequest	Indicates if the reply message should be returned.
UsrProf	The user profile.

See Also

Other Resources

Object Support Using Data Queues

IEIGDataQueueCtlEvents Notifications

The **IEIGDataQueueCtl** object of the Microsoft Data Queue ActiveX control also supports a set of events notifying a client application of connection status and error reporting. These events are handled by the client supporting several callback interfaces and setting these callbacks derived from the standard **IConnectionPointContainer** COM object.

The following **IEIGDataQueueCtlEvents** notification interface methods are supported by the Data Queue ActiveX control:

Event notifications	Comment
ConnectionStateChange	This event is fired when the state of a connection has changed. A ConnectionState parameter is passed to the client callback function that receives this event method call. This parameter is an **eigConnectionStateEnum** value representing the new state of the ConnectionState property.
Report Error	This event is fired when an error condition needs to be reported during non-blocking (asynchronous) functions. This event passes two parameters to the client callback function that receives this event method call. The first parameter is a long value representing an error code. The second parameter is a BSTR string containing a brief text description of the error.

See Also

Other Resources

Object Support Using Data Queues

IEIGDataQueueEvents Notifications

The **IEIGDataQueue** object of the Microsoft Data Queue ActiveX control also supports a set of events notifying a client application of when transfers are completed, requests are received, and error reporting. These events are handled by the client supporting several callback interfaces and setting these callbacks derived from the standard **IConnectionPointContainer** COM object.

The following **IEIGDataQueueEvents** notification interface methods are supported by the Data Queue ActiveX control:

Event notifications	**Comment**
ReportError2	This event is fired when an error condition needs to be reported during non-blocking (asynchronous) functions. This event passes two parameters to the client callback function that receives this event method call. The first parameter is a long value representing an error code. The second parameter is a BSTR string containing a brief text description of the error.
RequestReceived	This event is fired when a request is received.
SendComplete	This event is fired as an indication to the client that the requested transfer operation has completed.

See Also

Other Resources

Object Support Using Data Queues

Programming Considerations When Using the Data Queue ActiveX Control

The Microsoft® Data Queue ActiveX® control exposes a dual interface deriving from **IDispatch**. This provides support and flexibility to clients wishing to use the object. Clients that provide support for automation interfaces can use the **IDispatch** interface while more robust clients may use the custom interface. Using the custom interface offers the greatest execution speed.

The single-threading model is supported, allowing only single threads to access the objects safely.

Asynchronous read operations are not currently supported. The *BlockComplete* parameter of the GetQueueItem method must be set to a value of 0 (eigAnswerYes), indicating that the **GetQueueItem** operation should block until the completion status is known.

This section contains:

- Code Page Support Using Data Queues
- User Names and Passwords Using Data Queues
- Troubleshooting the Data Queue ActiveX Control

Code Page Support Using Data Queues

When using the Data Queue ActiveX control, the Host CCSID (character code set identifier) property should be configured to match the data as represented on the remote host computer. The Host CCSID parameter defaults to EBCDIC U.S./Canada (37) when using the Data Queue ActiveX control.

This section contains:

DBCS Code Page Support Using Data Queues

Support for Double-Byte Character String (DBCS) data is limited using the Data Queue ActiveX control. Conversions between DBCS and ANSI code pages are not supported. Conversions between DBCS and ISO code pages are not supported.

The DB2 GRAPHIC data types (GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC) are not supported. These DB2 data types support DBCS (not mixed) data. Mixed data types are supported using CHAR FOR MIXED DATA, VARCHAR FOR MIXED DATA, and LONGVARCHAR FOR MIXED DATA.

See Also

Other Resources

Code Page Support Using Data Queues

User Names and Passwords Using Data Queues

When connecting to host systems, most users must be authenticated by the remote system by passing a valid User ID and Password.

The AS/400 computer is case-sensitive with regard to user ID and password. The AS/400 only accepts a user ID and password in uppercase. The Microsoft Data Queue ActiveX control forces the user ID and password into uppercase when it knows that it is connecting to an AS/400 system.

See Also

Other Resources

Programming Considerations When Using the Data Queue ActiveX Control

Troubleshooting the Data Queue ActiveX Control

The Microsoft Data Queue ActiveX control supplied with Host Integration Server 2009 has the ability to trace DRDA data flows.

This tracing capability is accessible from the DB2 Network Librarytracing inside the Trace tool. This facility shows the same data as an APPC trace but without the control indicators (for example, What_Received). Socket errors are traced and the error codes can be looked up in Winsock2.h supplied with the Platform SDK.

The Data Queue ActiveX control can return the following types of errors:

- Errors from the remote hosts
- Microsoft Data Queue-specific errors
- Errors from the underlying DDM application requester network client

See Also

Other Resources

Programming Considerations When Using the Data Queue ActiveX Control

Using Data Design Tools

The Data Design Tools are a set of technologies exposed in Visual Studio that enable you to visually create database connections and queries, browse tables, view data, execute stored procedures, and access other database features. The following table describes the level of support the Server Explorer for Visual Studio provides for Host Integration Server 2009.

Provider	**Visual Studio Data Design Support**
ODBC Driver for DB2	None
OLE DB Provider for DB2	High
Managed Provider for DB2	Low
OLE DB Provider for AS/400 and VSAM	Low
Data Queues	None
File Transfer ActiveX controls	None

In This Section

Using Data Design Tools for the OLE DB Provider for DB2

Using Data Design Tools for the Managed Provider for DB2

Using Data Design Tools for the Microsoft OLE DB Provider for AS/400 and VSAM

Using Data Design Tools for the OLE DB Provider for DB2

The OLE DB Provider for DB2 is the provider that the Microsoft Visual Studio Data Design Tools most heavily support. The following list describes the different Data Design Tools available in Visual Studio, and how they interact with the OLE DB Provider for DB2.

- Database Designer

  With this visual tool, you can design and visualize a DB2 database to which you are connected. When designing a database, you can use the Database Designer to create, edit, or delete tables, columns, keys, indexes, relationships, and constraints. To visualize a database, you can create one or more diagrams illustrating some or all of the tables, columns, keys, and relationships.

- Table Designer

  With this visual tool, you can design and visualize a single table in a DB2 database to which you are connected.

- Query and View Designer

  With this visual tool, you can design a query, view, in-line function, or single-statement stored procedure.

- SQL Editor

  This integrated tool provides a variety of SQL text-editing features.

- Server Explorer

  With this Visual Explorer window, you can view and manipulate data links, database connections, and system resources on any server to which you have network access. Features include the ability to open data connections, log onto servers and display system resources, make database connections, and create data components that describe remote resources for your Visual Studio projects.

- Solution Explorer

  Provides an organized view of your projects and their files as well as ready access to the commands that pertain to them.

See Also
Other Resources
Using Data Design Tools

Using Data Design Tools for the Managed Provider for DB2

The Data Design Tools included in Microsoft® Visual Studio® .NET 2003 do not currently support the Managed Provider for DB2. However, you can partially prototype and design your data connections with System.Data and ADO.NET using the available tools.

To use the Data Design Tools for the Managed Provider for DB2

1. Use the Data Design Tools for the OLE DB Provider for DB2 to outline your data connection.

   The OLE DB Provider for DB2 and the Managed Provider for DB2 are similar enough that the generated scripts and objects can provide a starting point for your Managed Provider for DB2 connection.

2. Manually alter the autogenerated files for use with the Managed Provider for DB2.

See Also

Concepts

Using Data Design Tools for the OLE DB Provider for DB2

Other Resources

Managed Provider for DB2 Programmer's Guide

Using Data Design Tools for the Microsoft OLE DB Provider for AS/400 and VSAM

The Microsoft OLE DB Provider for AS/400 and VASM is a non-SQL command provider. As such, this provider does not integrate well with the Microsoft Visual Studio Data Design Tools. While you can create a data connection from the supplied menus in Visual Studio, very few of the generated scripts and objects are useful for a Host Integration Server 2009 application. Instead, it is recommended that you manually create the connections using the standard programming technologies.

See Also

Other Resources

OLE DB Provider for AS/400 and VSAM Programmer's Guide

Data Integration Security Guide

This section covers security issues that programmers working with the data integration features of Host Integration Server 2009 should understand.

In This Section

Managed Provider Security

The following topics discuss security as it applies to the Managed Provider for DB2 section of the Host Integration Server 2009 SDK.

About Managed Provider security for developers

A Managed Provider is a .NET Framework data provider used for connecting Host Integration Server 2009 applications to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET **DataSet**.

The Managed Provider for DB2 is built on the .NET Framework, and because of this it can take advantage of many of the security features that a managed environment provides. This section discusses security features and issues that you should be aware of when using the Managed Provider.

Threats and mitigations for the Managed Provider

Programmers who use the Managed Provider for DB2 should be aware of the following security practices and issues.

Protect Components Store Connection Information in Plain Text

Generic data access components, such as the Data Tools and Data Bound controls in Microsoft Visual Studio store sensitive DB2 connection data in plain text. This is done to grant third parties easy access to host data when using the control.

Storing connection data in plain text is a security concern because there is the potential for another application or user to read the active memory belonging to the control, and retrieve sensitive information from the plain text stored in the control. You can improve the security of your Host Integration Server application using one or more of the following techniques:

- Store any file known to contain connection information, such as a .UDL file, in a secure location.
- Design your applications so that you release and destroy any components known to store data in plain text as soon as the user is finished with the component.

OLE DB Provider for DB 2, AS/400, and VSAM Security

The following topics discuss security as it applies to the following sections of Host Integration Server 2009 SDK.

- OLE DB Provider for DB2
- OLE DB Provider for AS/400 and VASM

About OLE DB Provider Security for Developers

These Microsoft OLE DB providers enable users to access IBM DB2 from within an OLE-aware application. The object linking and embedding database (OLE DB) is a standard set of interfaces that provides heterogeneous access to disparate sources of information located anywhere—file systems, e-mail folders, and databases. The OLE DB Provider for DB2 combines the universal data access of OLE DB with the IBM Distributed Relational Database Architecture (DRDA).

This section discusses security features and issues that you should be aware of before programming the OLE DB Provider for DB2.

Single Sign-On

Enterprise Single Sign-On provides services that request and verify your credentials after you log on to the network, and use your credentials to determine the actions that you can perform based on your user rights. Single Sign-On is supported for the OLE DB provider for DB2.

For information on how you can configure Single Sign-On for the OLE DB Providers, see Connection.

Threats and mitigations for the OLE DB Provider

Programmers who use these features should be aware of the following security practices and issues.

The OLE DB Provider for DB2 stores host configuration data in OLE DB properties collection

The OLE DB Provider for DB2 stores host and PC configuration data in an OLE DB property collection, and then returns this data to the calling program. Your program should not pass this sensitive information on to the end user.

The OLE DB Provider for DB2 DTS and DQP consumers persist configuration data

Generic consumers such as DTS, DQP, Replication, and OLAP persist sensitive host configuration information in the following insecure ways:

- Most SQL Server consumers use the SQL Server Repository to store data source information.
- DTS can persist DTS packages (which include data source configuration info) as files and Visual Basic programs (plain text).

To minimize the risks mentioned above, generic consumers should encrypt sensitive OLE DB connection information using the crypto API (DP, data protection API).

The OLE DB Provider for AS/400 and VSAM stores host configuration data in an OLE DB properties collection

The OLE DB Provider for AS/400 and VSAM stores host and computer configuration data in an OLE DB property collection, and then returns this data to the calling program. Your program should not pass this sensitive information on to the end user.

Use provider-specific connection pooling

It is a best practice to use the provider-specific connection pooling in Host Integration Server rather than the system provided resource pooling provided by the OLE DB Provider for DB2. This avoids possible security risks in the OLE DB Provider for DB2.

The OLE DB Provider for DB2 DTS and DQP consumers persist configuration data

Generic consumers such as DTS, DQP, Replication and OLAP stores sensitive host configuration information in the following insecure ways:

- Most SQL Server consumers utilize the SQL Server Repository to store data source information.
- DTS can store DTS packages (which include data-source configuration info) as files and Visual Basic programs (plain text).

To minimize the risks mentioned above, Generic consumers should encrypt sensitive OLE DB connection information using the crypto API (DP, data protection API).

Host File Transfer Object Security

The Microsoft Host File Transfer ActiveX control provides the ability to transfer files between a local computer and an MVS, OS/390, AS/400, or AS/36 host system. Host Integration Server 2009 provides this service through a single ActiveX control that depends on other core Host Integration Server DLLs.

This section discusses security features and issues that you should be aware of before you use with the Host File Transfer object in your programs.

Threats and mitigations for the Host File Transfer object

Programmers who use these features should be aware of the following security practices and issues.

Validate input parameters for the Host File Transfer control

You should validate all input parameters before passing them on to the Host File Transfer control.

You should verify all values received from end users before passing values (such as filenames) to the ActiveX control.

The SAM and VSAM data sets, as well as PDS/PDSE members, must be referenced in the host column description (HCD) file the same way as they are referenced in the command text of the OLE DB command object, using the following command text syntax:

```
EXEC OPEN FileName
```

where FileName represents one of the following host file naming conventions:

- Host file type File naming convention
- VSAM Data Sets:

  ```
  DATASETNAME.FILENAME
  ```

- Partitioned Data Sets:

  ```
  DATASETNAME.FILENAME(MEMBER)
  ```

- OS/400 Files:

  ```
  LIBRARY/FILE
  ```

- OS/400 Files:

  ```
  LIBRARY/FILENAME
  ```

- OS/400 File Members:

  ```
  LIBRARY/FILE(MEMBER)
  ```

- OS/400 File Members:

  ```
  LIBRARY.FILENAME(MEMBER)
  ```

Note that if a member of a library contains a dot in the member name, the member name must be surrounded by double quotes. For example, if the member name is NAMES.DAT, the proper syntax used to open a rowset using command text is as

follows:

```
EXEC OPEN LIBRARY/FILE("NAMES.DAT")
```

Please note that you must utilize the full path to the mainframe data set. In the example below, there are two path elements and one name element to describe the target data set:

```
XXXXX.XXX.XXX
```

Whenever you allocate a data set, it is given a unique name composed of one or more segments. Each segment of a data set name is joined by periods and represents a level of qualification. For example, the following data set has four segments that comprise the fully qualified data set name.

```
WNW999.DEMO.KSDS.TITLES
```

The high-level qualifier is WNW999. The low-level qualifier is TITLES. Each segment can be from 1-8 characters in length (first character must be alphabetic; remainder can be alphanumeric or hyphens). The data set name must be no more than 44 characters in length and contain no more than 22 segments.

Store the HCD files for the Host File Transfer control in a secure location

File Transfer reads host column description (HCD) files at run time as a means to support data conversion from host data types to OLE DB data types. You should store HCD files in a secure location that only the application can access at run time. This ensures that a malicious user may not modify the file to submit embedded commands.

Network Integration Programmer's Guide

This section describes how to create applications in a Systems Network Architecture (SNA) environment.

For API references and other technical information about network integration, see the Network Integration Programmer's Reference section of the SDK.

For sample code that illustrates network integration, see the Network Integration Samples section of the SDK.

For additional information about network integration, see the Network Integration User's Guide section of the Operations guide.

In This Section

- APPC Programmer's Guide
- CPI-C Programmer's Guide
- LUA Programmer's Guide
- 3270 Emulation Programmer's Guide
- SNA Internationalization Programmer's Guide
- SNA Print Server Data Filter Programmer's Guide
- SNADIS Programmer's Guide
- Network Integration Security Guides
- Session Integrator Programmer's Guide
- Client-Based BizTalk Adapter for WebSphere MQ Programmer's Guide

APPC Programmer's Guide

This section of the Host Integration Server 2009 Developer's Guide provides information about using the Advanced Program-to-Program Communications (APPC) in a distributed processing environment.

For API references and other technical information about APPC, see APPC Programmer's Reference.

For sample code using APPC, see APPC Samples.

This section contains:

- About the APPC Guide
- Introduction to APPC
- About Transaction Programs
- Windows CSV Overview
- Support for APPC Automatic Logon

APPC Guide

This section provides information required to develop C-language applications that use Advanced Program-to-Program Communications (APPC) to exchange data in a Systems Network Architecture (SNA) environment.

This section is intended for the developer writing applications that use Common Programming Interface for Communications (CPI-C) to exchange data. It provides conceptual information and detailed reference information.

To use this section effectively, you should be familiar with:

- Microsoft Host Integration Server 2009
- Microsoft Windows Server 2003 or Windows 2000
- SNA concepts

This section contains:

Operating Systems Support for APPC Development

Microsoft Host Integration Server 2009 supports the development of Advanced Program-to-Program Communications (APPC) applications for Microsoft Windows Server 2003 and Microsoft Windows 2000.

Finding Further Information about APPC

This section does not describe the products, architectures, or standards developed by other companies or organizations.

For more information about SNA and about 3270 information display systems, see the following manuals:

- *IBM 3270 Information Display System: 3274 Control Unit Description and Programmers Guide*
- *IBM 3270 Information Display System: Color and Programmed Symbols*
- *IBM 3270 Information Display System: 3274 Control Unit Display Station: Operators Guide*
- *IBM Systems Network Architecture: Technical Overview*
- *IBM Systems Network Architecture: Concepts and Products*
- *IBM Advanced Communications Function Products Installation Guide*
- *IBM Installation and Resource Definition*
- *IBM 9370 LAN Token Ring Support*
- *IBM SNA Format and Protocol Reference Manual: Architectural Logic*

For background information about logical unit (LU) 6.2, APPC, or the Common Programming Interface for Communications (CPI-C), see the following manuals:

- *IBM Systems Network Architecture: Introduction to APPC*
- *IBM Systems Network Architecture: Transaction Programmers Reference Manual for LU Type 6.2*
- *IBM SNA: Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM SNA: Formats*
- *IBM SNA: Technical Overview*
- *IBM SNA: ACF/VTAM Programming for LU Type 6.2*

Introduction to APPC

This section introduces the fundamental concepts of Advanced Program-to-Program Communications (APPC) in a distributed processing environment. These concepts include the following:

- APPC verbs
- Microsoft® Windows® APPC extensions
- Using APPC verbs in C programs
- Operating system considerations

Detailed descriptions of APPC verbs are provided in:

- APPC Management Verbs
- APPC TP Verbs
- APPC Conversation Verbs

APPC is an application programming interface (API) that enables peer-to-peer communications in a Systems Network Architecture (SNA) environment. Through APPC, programs distributed across a network can work together, communicating with each other and exchanging data, to accomplish a single processing task such as querying a remote database, copying a remote file, or sending and receiving electronic mail.

This section contains:

- APPC Verb Overview
- APPC Verb Summary
- Windows APPC Overview
- Using APPC Verbs in C Programs
- Windows Server 2003, Windows XP, and Windows 2000 Considerations

APPC Verb Overview

APPC verbs fall into three categories: management, transaction program (TP), and conversation.

Management Verbs

Management verbs provide management functions. They are:

ACTIVATE_SESSION

CNOS

DEACTIVATE_SESSION

DISPLAY

TP Verbs

TP verbs start and end TPs, and get and set TP properties. They are:

GET_TP_PROPERTIES

SET_TP_PROPERTIES

TP_ENDED

TP_STARTED

Conversation Verbs

Conversation verbs enable TPs to allocate and deallocate conversations, send and receive data, and change conversation states. The conversation verbs are listed in the following table.

Conversation verbs fall into two groups: mapped conversation verbs and basic conversation verbs. The mapped conversation is intended for programs that use the conversation directly. The basic conversation is intended for more complex programs that provide services to other users. In typical situations, end-user TPs use mapped conversations and service TPs use basic conversations.

Mapped conversation verbs can only be issued by a TP in mapped conversations, while basic conversation verbs are reserved for basic conversations. There is one exception to this rule: ALLOCATE can be used to start either a basic or a mapped conversation.

Mapped conversation verbs	**Basic conversation verbs**
MC_ALLOCATE	ALLOCATE
MC_CONFIRM	CONFIRM
MC_CONFIRMED	CONFIRMED
MC_DEALLOCATE	DEALLOCATE
MC_FLUSH	FLUSH
MC_GET_ATTRIBUTES	GET_ATTRIBUTES
GET_LU_STATUS	GET_LU_STATUS
GET_STATE	GET_STATE
GET_TYPE	GET_TYPE
MC_POST_ON_RECEIPT	POST_ON_RECEIPT

MC_PREPARE_TO_RECEIVE	PREPARE_TO_RECEIVE
RECEIVE_ALLOCATE	RECEIVE_ALLOCATE
MC_RECEIVE_AND_POST	RECEIVE_AND_POST
MC_RECEIVE_AND_WAIT	RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE	RECEIVE_IMMEDIATE
MC_RECEIVE_LOG_DATA	RECEIVE_LOG_DATA
MC_REQUEST_TO_SEND	REQUEST_TO_SEND
MC_SEND_CONVERSATION	SEND_CONVERSATION
MC_SEND_DATA	SEND_DATA
MC_SEND_ERROR	SEND_ERROR
MC_TEST_RTS	TEST_RTS

Mapped and basic verbs have the same functions in their respective types of conversation. For example, **MC_CONFIRM** performs the same function in a mapped conversation that **CONFIRM** performs in a basic conversation.

APPC Verb Summary

This section briefly describes each APPC verb, grouped by function.

Verbs for Starting Conversations
ALLOCATEor MC_ALLOCATE

Issued by the local transaction program (TP). This verb allocates a session between the local logical unit (LU) and a partner LU, and establishes a conversation between the local TP and the partner TP.

ALLOCATE can establish either a basic or a mapped conversation. **MC_ALLOCATE** can start only a mapped conversation. After the conversation is allocated, APPC uses this verb to return a conversation identifier (**conv_id**).

RECEIVE_ALLOCATE

Issued by the partner TP. This verb confirms that the partner TP is ready to begin a conversation with the local TP that issued **ALLOCATE** or **MC_ALLOCATE**. Upon successful execution, this verb returns a TP identifier (**tp_id**) for the partner TP and the **conv_id**.

TP_STARTED

Issued by the local TP. This verb notifies APPC that the local TP is starting. Upon successful execution, this verb returns a **tp_id** for the local TP.

Verbs for Sending Data
CONFIRMor MC_CONFIRM

Sends the contents of the local LU's send buffer and a confirmation request to the partner TP.

FLUSHor MC_FLUSH

Flushes the local LU's send buffer, sending the contents of the buffer to the partner LU and TP. If the send buffer is empty, no action takes place.

PREPARE_TO_RECEIVEor MC_PREPARE_TO_RECEIVE

Changes the state of the conversation from SEND to RECEIVE. Before changing the conversation state, this verb performs the equivalent of **FLUSH**, **MC_FLUSH**, **CONFIRM**, or **MC_CONFIRM**. After this verb has successfully executed, the local TP can receive data.

REQUEST_TO_SENDor MC_REQUEST_TO_SEND

Informs the partner TP that the local TP wants to send data. The local TP must wait until the partner TP issues **PREPARE_TO_RECEIVE**, **MC_PREPARE_TO_RECEIVE**, **RECEIVE_AND_WAIT**, or **MC_RECEIVE_AND_WAIT**, and the conversation state changes to RECEIVE for the partner TP, before the local TP begins sending data.

SEND_DATAor MC_SEND_DATA

Puts data in the local LU's send buffer for transmission to the partner TP.

The data collected in the local LU's send buffer is transmitted to the partner LU and partner TP when one of the following occurs:

- The send buffer fills up.
- The local TP issues **FLUSH**, **MC_FLUSH**, **CONFIRM**, **MC_CONFIRM**, **DEALLOCATE**, **MC_DEALLOCATE**, or another verb that flushes the local LU's send buffer.

Verbs for Receiving Data
POST_ON_RECEIPTor MC_POST_ON_RECEIPT

Issuing this verb allows the application to register to receive a notification when data or status arrives at the local LU without actually receiving it at the same time. This verb can only be issued while in RECEIVE state and it never causes a change in conversation state.

When the TP issues this verb, APPC returns control to the TP immediately. When the specified conditions are satisfied, the Win32® event specified as a parameter is signaled and the verb completes. Then the TP looks at the return code in the verb control block to determine whether or not any data or status notification has arrived at the local LU and issues a

RECEIVE_IMMEDIATE or **RECEIVE_AND_WAIT** verb to actually receive the data or status notification.

RECEIVE_AND_POSTor MC_RECEIVE_AND_POST

Issuing this verb while the conversation is in RECEIVE state changes the conversation state to PENDING_POST and causes the local TP to receive data asynchronously. This allows the local TP to proceed with processing while data is still arriving at the local LU.

Issuing this verb while the conversation is in SEND state flushes the LU's send buffer and changes the conversation state to PENDING_POST. The local TP then begins to receive data asynchronously.

RECEIVE_AND_WAITor MC_RECEIVE_AND_WAIT

Issuing this verb while the conversation is in RECEIVE state causes the local TP to receive any data that is currently available from the partner TP. If no data is available, the local TP waits for data to arrive.

Issuing this verb while the conversation is in SEND state flushes the LU's send buffer and changes the conversation state to RECEIVE. The local TP then begins to receive data.

RECEIVE_IMMEDIATEor MC_RECEIVE_IMMEDIATE

Receives any data that is currently available from the partner TP. If no data is available, the local TP does not wait.

TEST_RTSor MC_TEST_RTS

Determines whether a **REQUEST_TO_SEND** or **MC_REQUEST_TO_SEND** or notification has been received.

Verbs for Confirming Data or Reporting Errors
CONFIRMEDor MC_CONFIRMED

Replies to a confirmation request from the partner TP. It informs the partner TP that the local TP has received and processed the data without error.

RECEIVE_LOG_DATAor MC_RECEIVE_LOG_DATA

Issuing this verb allows the user to register to receive the log data associated with an inbound Function Management Header 7 (FMH7) error report. The verb passes a buffer to APPC, and any log data received is placed in that buffer. APPC continues to use this buffer as successive FMH7s arrive until it is provided with another buffer (that is, until the TP issues another **RECEIVE_LOG_DATA** or **MC_RECEIVE_LOG_DATA** specifying a different buffer or no buffer at all).

SEND_CONVERSATIONor MC_SEND_CONVERSATION

Issued by the invoking TP, this verb allocates a session between the local LU and partner LU, sends data on the session, and then deallocates the session.

SEND_ERRORor MC_SEND_ERROR

Notifies the partner TP that the local TP has encountered an application-level error.

Verbs for Getting and Setting Information
GET_ATTRIBUTESor MC_GET_ATTRIBUTES

Used by a TP to get the attributes of the conversation.

GET_LU_STATUS

Used to report the status of a particular remote LU.

GET_STATE

Used by a TP to interrogate the state of a particular conversation.

GET_TP_PROPERTIES

Returns attributes of the TP and the current transaction.

GET_TYPE

Used by a TP to determine the conversation type (basic or mapped) of a particular conversation. With this information, the TP can decide whether to issue basic or mapped conversation verbs.

SET_TP_PROPERTIES

Used to set the attributes of the TP and the current transaction.

Verbs that Provide Management Functions

ACTIVATE_SESSION

Activates a session between the local LU and a specified partner LU, using a specified mode.

CNOS(Change Number of Sessions)

Establishes APPC LU 6.2 session limits.

DEACTIVATE_SESSION

Deactivates a particular session, or all sessions on a particular mode.

DISPLAY

Returns configuration information and current operating values for the SNA node.

Verbs for Ending Conversations

DEALLOCATEor MC_DEALLOCATE

Deallocates a conversation between two TPs. Before deallocating the conversation, this verb performs the equivalent of **FLUSH**, **MC_FLUSH**, **CONFIRM**, or **MC_CONFIRM**.

TP_ENDED

Issued by both the local and partner TPs. It notifies APPC that the TP is ending. Issuing this verb also terminates any active conversations.

Windows APPC Overview

The information provided in this guide is source code and executable code compatible with the following implementations of APPC:

- APPC applications based on Host Integration Server 2009 residing on the server or on a client. These applications run on Microsoft Windows Server, Windows XP, and Windows 2000.

Programs written to use this implementation of APPC can exchange data with programs written to use other implementations of APPC that adhere to the SNA LU 6.2 architecture.

The use of the Windows APPC interface on Windows Server 2003 and Windows 2000 causes additional threads to be created within the calling process. These other threads perform interprocess communication with the Host Integration Server 2009 or SNA service over the LAN interface that the client is configured to use (TCP/IP, IPX/SPX, or named pipes, for example).

If an application using Windows APPC is running on Windows Server 2003 and Windows 2000, stopping the SNABASE service causes the application to be unloaded from memory.

In This Section

Windows APPC Asynchronous Support

APPC Verbs and Windows Extensions

Windows APPC Asynchronous Support

A program that issues a call and does not regain control until the call completes cannot perform any other operations. This type of operation, referred to as blocking, is not suited to a server application designed to handle multiple requests from many clients. Asynchronous call completion returns the initial call immediately so the application can continue with other processes.

Host Integration Server 2009 uses the **RegisterWindowsMessage** function for asynchronous support for APPC applications. With "WinAsyncAPPC" as the input string, an application passes a window handle by which it can be notified of verb completion. The application then issues the verb. When the verb completes, a message is posted to the window handle that was passed, notifying the application that the verb is complete.

With the exception of asynchronous RECEIVE_AND_WAIT, MC_RECEIVE_AND_WAIT, RECEIVE_AND_POST, and MC_RECEIVE_AND_POST, which can issue certain other verbs while pending, a conversation can have only one incomplete operation at any time.

APPC Verbs and Windows Extensions

This topic describes the APPC verbs and Windows extensions that are supported by Host Integration Server 2009:

APPC Verbs

The following APPC verb descriptions contain important features and should be read before using this version of Windows APPC.

ALLOCATEor **MC_ALLOCATE**

Issued by the invoking transaction program (TP), this verb allocates a session between the local logical unit (LU) and partner LU and (in conjunction with **RECEIVE_ALLOCATE**) establishes a conversation between the invoking TP and the invokable TP. After this verb executes successfully, APPC generates a conversation identifier (**conv_id**). The **conv_id** is a required parameter for all other APPC conversation verbs.

For a user or group using TPs, 5250 emulators, or APPC applications, you can assign default local and remote LUs. In this case, the field for LU alias is left blank or null and the default LUs are accessed when the user or group member starts an APPC program. For more information about using default LUs, see the **Network Integration**section of the Microsoft Host Integration Server Help.

RECEIVE_ALLOCATE

Issued by the invokable TP to confirm that it is ready to begin a conversation with the invoking TP that issued **ALLOCATE** or **MC_ALLOCATE**. This must be the first APPC verb issued by the invokable TP. The initial state is RESET. If the verb executes successfully (**primary_rc** is AP_OK), the state changes to RECEIVE.

RECEIVE_AND_POSTor **MC_RECEIVE_AND_POST**

Receives application data and status information asynchronously. This enables the local TP to proceed with processing while data is still arriving at the local LU. RECEIVE_AND_POST and MC_RECEIVE_AND_POST are only supported by the Windows 2000 operating system.

While an asynchronous RECEIVE_AND_POST or MC_RECEIVE_AND_POST is outstanding, the following verbs can be issued:

REQUEST_TO_SEND or MC_REQUEST_TO_SEND

GET_TYPE

GET_ATTRIBUTES or MC_GET_ATTRIBUTES

TEST_RTS or MC_TEST_RTS

DEALLOCATE

SEND_ERROR or MC_SEND_ERROR

TP_ENDED

RECEIVE_AND_WAITor **MC_RECEIVE_AND_WAIT**

Receives any data that is currently available from the partner TP. If no data is currently available, the local TP waits for data to arrive.

RECEIVE_AND_WAIT and MC_RECEIVE_AND_WAIT have been altered to act like RECEIVE_AND_POST and MC_RECEIVE_AND_POST. While an asynchronous RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT is outstanding, the following verbs can be issued:

REQUEST_TO_SEND or MC_REQUEST_TO_SEND

GET_TYPE

GET_ATTRIBUTES or MC_GET_ATTRIBUTES

TEST_RTS or MC_TEST_RTS

DEALLOCATE

SEND_ERROR or MC_SEND_ERROR

TP_ENDED

TP_STARTED

Issued by the invoking TP, this verb notifies APPC that the TP is starting. For a user or group using TPs, 5250 emulators, or APPC applications, you can assign default local and remote APPC LUs. These default LUs are accessed when the user or group member starts an APPC program (a TP, 5250 emulator, or APPC application) and the program does not specify LU aliases. For more information about using default LUs, see **Network Integration Help**.

Limits

Host Integration Server 2009 permits one outstanding Windows APPC asynchronous call per connection and one blocking verb per thread. For example:

```
void ProcessVerbCompletion (WPARAM wParam, LPARAM lParam)
{
    int i;
    for (i = 0; i < nPendingVerbs; i++)
        if (pPendingVerbs[i].hAsync == wParam)
            ProcessVCB( (LPVCB) lParam);
}    . . .
LRESULT CALLBACK SampleWndProc ( ... )
{
    if (msg == uAsyncAPPC) {
        ProcessVerbCompletion(wParam; lParam);
    }
        else switch (msg) {
        case WM_USER:
            if (hAsync = WinAsyncAPPC(hwnd, &vcb))
               pPendingVerbs [nPendingVerbs++].hAsync = hAsync;
            break;
    }
}
WinMain ( ... )
{
    if ( ( WinAPPCStartup ( ...) = = FALSE ) {
    return FALSE ;
    }
    uAsyncAPPC = RegisterWindowsMessage ("WinAsyncAPPC") ;
    while (GetMessage ( ...)  )  {
    ...
    WinAPPCCleanup ( ... )
}
```

Note

The exceptions to the rule of one outstanding asynchronous call are **RECEIVE_AND_POST**, **MC_RECEIVE_AND_POST**, **RECEIVE_AND_WAIT**, and **MC_RECEIVE_AND_WAIT**. While these verbs are outstanding, certain other verbs can also be called.

Using APPC Verbs in C Programs

This implementation of APPC is available for programs written in Microsoft® C version 5.1 or later. A C program calls APPC through the external function **APPC**. For compatibility with previous versions of Microsoft C, the external function **APPC_C** is also supported.

Note
Compilers other than the Microsoft C compiler can also be used to build applications using this implementation of APPC.

This section contains:

- Verb Control Block
- APPC Definition
- Issuing an APPC Verb

Verb Control Block

The only parameter passed to the **APPC** function is the address of a verb control block (VCB). The VCB is a structure made up of variables that:

- Identify the APPC verb to be executed.
- Supply information to be used by the verb.
- Contain information returned by the verb when execution is complete.

Each APPC verb has its own VCB structure, which is declared in the WINAPPC.H header file. For compatibility with earlier versions, the APPC_C.H header file is also supported.

The WINAPPC.H file is supplied as part of the Host Integration Server 2009 Software Development Kit (SDK).

APPC Definition

The prototype definitions of the **APPC** function are as follows:

Syntax

```
void WINAPI APPC(long);
HANDLE WINAPI WinAsyncAPPC (hWnd, LPAPPC);
```

Remarks

The verb control block (VCB) address parameter, a 32-bit pointer, is declared as a long integer and thus requires casting from a pointer to a long integer.

Issuing an APPC Verb

The following procedure is required to issue a blocking APPC verb. In the sample code, the verb issued is **MC_SEND_DATA**.

To issue a blocking APPC verb

1. Create a structure variable from the verb control block (VCB) structure that applies to the APPC verb to be issued.

```
#include <winappc.h>
    .
    .
struct mc_send_data  mcsend;
The VCB structures are declared in WINAPPC.H; one of these structures is:
mc_send_data
```

2. Clear (set to zero) the variables within the VCB structure.

```
memset( mcsend, '\0', sizeof( mcsend ) );
```

3. Assign values to the VCB variables that supply information to APPC.

```
mcsend.opcode = AP_M_SEND_DATA;
mcsend.opext = AP_MAPPED_CONVERSATION;
memcpy( mcsend.tp_id, tp_id, sizeof( tp_id ) );
mcsend.conv_id = conv_id;
mcsend.dlen = datalen;
mcsend.dptr = sharebufptr;
```

   The values AP_MAPPED_CONVERSATION and AP_M_SEND_DATA are symbolic constants representing integers. These constants are defined in WINAPPC.H.

4. Invoke the **APPC** function. The only parameter is a pointer to the address of the structure containing the VCB for the desired verb.

```
APPC ( ( long ) (void FAR * ) &mcsend );
```

   Use WinAsyncAPPC if you are running the application under Windows version 3.*x*.

   To call **WinAsyncAPPC**:

```
HANDLE WINAPI WinAsyncAPPC (hWnd, 1pVCB)
```

   When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinAsyncAPPC" as the input string.

5. Use the variables that were returned by APPC.

```
if( mcsend.primary_rc != AP_OK )
/* Do error routine */
    .
    .
    .
```

Windows Server 2003, Windows XP, and Windows 2000 Considerations

This topic summarizes information about developing transaction programs (TPs) using APPC on the following operating systems:

- Microsoft Windows Server 2003
- Microsoft Windows XP Professional
- Microsoft Windows 2000

Byte ordering

The values of constants defined in WINAPPC.H and WINCSV.H are dependent on the byte ordering of the hardware used. Macros are used to set the constants to the correct value.

By default, Intel little-endian byte ordering is used, with the low byte of a 16-bit value followed by the high byte. However, when defining inline macros, the NON_INTEL_BYTE_ORDER macro used in WINAPPC.H and WINCSV.H will not reverse (flip) the byte order for constants. Non-constant input parameters in verb control blocks (VCBs) (such as lengths, pointers, and so on) are always in the native format.

For example, the primary return code of AP_PARAMETER_CHECK is defined to have a value of 0x0001. Depending on the environment (byte ordering), the constant AP_PARAMETER_CHECK may or may not be 0x0001. Some formats define the value as it appears in memory; others define it as a 2-byte variable. Because you cannot assume that the application always uses provided constants rather than hardwired values, you can define a macro to swap the bytes. The following is an example of using the macro:

```
/* when NON_INTEL_BYTE_ORDER is specified, the APPC_FLIPI macro defined in WINAPPC.H macr
o becomes */
#define APPC_FLIPI(x)    (x)

/* otherwise this macro flips bytes by defining */
#define APPC_FLIPI(X) APPC_MAKUS(APPC_HI_UC(X),APPC_LO_UC(X))

/* the AP_PARAMETER_CHECK macro is now defined using the APPC_FLIPI macro */
#define AP_PARAMETER_CHECK APPC_FLIPI (0X0001)      /* X '0001' */
```

Events

To receive data asynchronously, an event handle is passed in the semaphore field of the VCB. This event must be in the nonsignaled state when passed to APPC, and the handle must have EVENT_MODIFY_STATE access to the event.

Library names

In order to support the coexistence of Win16 and Win32 API libraries on the same computer, the Win32 DLL names have been changed.

Old DLL names	New DLL names
WINAPPC.DLL	WAPPC32.DLL
WINCSV.DLL	WINCSV32.DLL

The new DLL names should be used for Win32-based applications that are intended to run only on Host Integration Server 2009.

Limits

For Windows Server 2003 and Windows 2000, the number of simultaneous common service verbs (CSVs) allowed per process is 64. Only one of these verbs per thread can be synchronous (blocking).

Using APPC, the maximum number of simultaneous conversations per process is 15,000. Each process supports up to 15,000 simultaneous TPs.

Multiple threads

A TP can have multiple threads that issue verbs. Windows APPC makes provisions for multithreaded Windows-based processes. A process contains one or more threads of execution. All references to threads refer to actual threads in the multithreaded Windows Server 2003 and Windows 2000 environments.

With the exception of RECEIVE_AND_POST, MC_RECEIVE_AND_POST, RECEIVE_AND_WAIT, and MC_RECEIVE_AND_WAIT, only one conversation verb can be outstanding at a time on any conversation; however, other verbs can be issued for other conversations. This guideline also applies to TP verbs and TPs. Although multiple TP verbs can be issued, only one TP verb can be outstanding at a time on a TP. This applies to both multithreaded applications and single-threaded applications that use asynchronous calls.

Packing

For performance considerations, the VCBs are not packed. VCB structure member elements after the first element are aligned on either the size of the member type or DWORD (4-byte) boundaries, whichever is smaller. As a result, DWORDs are aligned on DWORD boundaries, WORDs are aligned on WORD boundaries, and BYTEs are aligned on BYTE boundaries. This means, for example, that there is a 2-byte gap between the primary and secondary return codes. Therefore, the elements in a VCB should only be accessed using the structures provided.

This option for structure and union member alignment is the default behavior for Microsoft C/C++ compilers. For compatibility with the supplied logical unit application (LUA) libraries, make sure to use an equivalent structure and union member packing option when using other C/C++ compilers or when explicitly specifying a structure alignment option when using Microsoft compilers.

Registering and deregistering applications

All Windows APPC applications must call WinAPPCStartup at the beginning of the session to register the application and WinAPPCCleanup at the end of the session to deregister the application.

All Windows CSV applications must call the Windows SNA extension WinCSVStartup at the beginning of the session to register the application and WinCSVCleanup to deregister the application when the session is finished.

Run-time linking

For a TP to be dynamically linked to APPC at run time, the TP must issue the following calls:

- **LoadLibrary** to load the dynamic-link libraries WINAPPC.DLL or WAPPC32.DLL.
- **GetProcAddress** to specify APPC on all the desired entry points to the DLL such as **APPC**, **WinAsyncAPPC**, **WinAPPCStartup**, and **WinAPPCCleanup**.

For a TP to be dynamically linked to CSV at run time, the TP must issue the following calls:

- **LoadLibrary** to load WINCSV.DLL or WINCSV32.DLL, the dynamic-link libraries for Windows CSV.
- **GetProcAddress** to specify CSV on all the desired entry points to the DLL such as **ACSSVC**, **WinAsyncCSV**, **WinCSVStartup**, and **WinCSVCleanup**.

The TP must issue the **FreeLibrary** call when the APPC or CSV library is no longer required.

Yielding to other components

Because the Windows Server 2003 and Windows 2000 environments are multithreaded, there is no need to yield to other components.

Transaction Programs Overview

A processing task accomplished by programs using Advanced Program-to-Program Communications (APPC) is called a transaction. Consequently, programs that use APPC are called transaction programs (TP)s. These programs communicate as peers, on an equal (rather than hierarchical) basis. The TPs use APPC verbs to exchange status information and application data. Each TP uses APPC verbs to supply parameters to APPC, which performs the desired function and returns parameters to the TP.

TPs distributed across a local or wide area network perform distributed transaction processing.

This section describes how to write TPs and how to configure the systems on which TPs run. The topics in this section cover the following general areas:

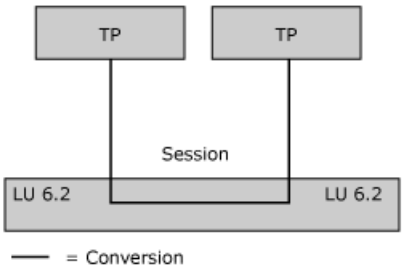
- Understanding fundamental concepts related to TPs
- Designing and coding TPs
- Configuring registry and environment variables for invokable TPs
- Configuring Host Integration Server 2009 to work with your TPs
- Sync Point Level 2 support

This section contains:

Communication between TPs

Various hardware and software elements in the SNA environment are required for two transaction programs (TPs) to communicate with each other. The following figure shows several fundamental elements.

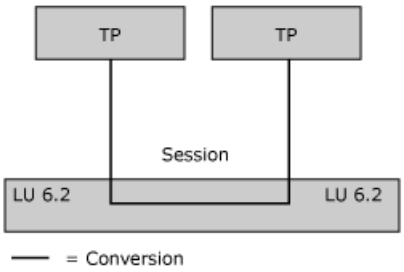
Fundamental communications elements between type 6.2 logical units



Each TP is associated with a logical unit (LU) of type 6.2. The LU allows the TP to access the network. Several TPs can be associated with the same LU.

A partner TP can invoke another TP, which, in turn, invokes another TP, and so on. In the following figure, TP A invokes TP B and TP B invokes TP C.

TP A invoking TP B and TP B invoking TP C.



This section contains:

- Fundamental Terms for TPs and LUs
- Sample TPs Illustrating Fundamental Concepts
- Configuring and Controlling TPs
- Creating TPs and Their Supporting Configuration

Fundamental Terms for TPs and LUs

The following terms describe some fundamental characteristics of transaction programs (TPs) communicating through logical units (LUs):

asynchronous verb
An APPC verb for which the initial function call returns immediately, so that the normal operation of the program is not blocked while processing of the verb completes. For more information, see Receiving Data Asynchronously.

basic conversation
A type of conversation more complex than a mapped conversation and generally used by service TPs (SNA-based programs that provide services to other programs). For more information, see Basic and Mapped Conversations Compared.

conversation
The interaction between TPs carrying out a specific task. Each conversation requires an LU-LU session. A TP can be involved in several conversations simultaneously, as shown with TP B in Communication between TPs.

invokable TP
A TP that can be invoked by another TP. Invokable TPs are usually server-type applications; that is, they work in the same general way that an application such as CICS works. Parameters for an invokable TP are configured through registry or environment variables.

There are several types of invokable TPs:

operator-started invokable TP

A TP that is started manually in preparation for being invoked.

autostarted invokable TP

A TP that is automatically started by APPC when invoked.

queued TP

A TP that, when invoked multiple times, loads once and then queues up subsequent requests to be dealt with one at a time. All operator-started TPs and some autostarted TPs are queued.

nonqueued TP

A TP loaded multiple times, once for every time it is invoked. Some autostarted TPs are nonqueued but no operator-started TPs are nonqueued.

For more information, see Invokable TPs.

invoking TP
A TP that can invoke (that is, initiate a conversation with) other TPs. Invoking TPs are usually client-type applications; that is, they work in the same general way that an emulator works. For more information, see Invoking TPs.

local LUand **local TP**
An LU and TP working together, when viewed as the "home base" for a particular conversation. From this viewpoint, some other LU and TP are seen as the "partner" or "remote" LU and TP.

LU alias
The string that identifies an LU to a TP. The alias can be up to eight characters long.

LU-LU session
The communication between two LUs over a specific connection for a specific amount of time. An LU-LU session is needed for two TPs to interact. One session can be used serially by many pairs of TPs.

An LU 6.2 can have multiple sessions (two or more concurrent sessions with different partner LUs) and parallel sessions (two or more concurrent sessions with the same partner LU).

LUs as well as LU-LU pairs and modes are configured using the SNA Manager on Host Integration Server.

mapped conversation
A type of conversation simpler than a basic conversation and generally used by application TPs (programs that accomplish

tasks for end users). The characters **MC_** at the beginning of a verb stand for mapped conversation. For more information, see Basic and Mapped Conversations Compared.

partner LUand **partner TP**, or **remote LU**and **remote TP**
An LU and TP working together, when viewed as being at the far end of a particular conversation.

synchronous verb
An APPC verb that blocks further program operations until the processing of the verb is complete.

Sample TPs Illustrating Fundamental Concepts

A set of sample transaction programs (TPs) is provided on the Host Integration Server CD-ROM in the \SDK\SAMPLES directory. Included with the sample code in the \SDK\SAMPLES\SNA\TPSETUP directory on the Host Integration Server CD-ROM is TPSETUP, a program that simplifies the setting of registry or environment variables needed by autostarted invokable TPs. Without an interface like that provided by TPSETUP, configuring such variables can be complicated and error-prone. Therefore, it is recommended that you use code like TPSETUP in installation programs for autostarted invokable TPs.

The source code for TPSETUP (INSTALL.C) can be compiled to work in the Microsoft Windows Server™ 2003 and Windows® 2000 environment.

For information about TPSETUP and about the sample TPs, see APPC Samples.

Configuring and Controlling TPs

The following table shows how the characteristics of the transaction programs (TPs) and selection of the logical units (LUs) for a conversation are controlled.

Characteristic	How controlled
Type of verb: synchronous or asynchronous	Written into the code. Synchronous verbs use blocking calls; asynchronous verbs avoid blocking calls. See Receiving Data Asynchronously and WinAsyncAPPC.
Type of conversation: basic or mapped	Written into the code. The **MC_** prefix is used on verbs in mapped conversations and omitted on verbs in basic conversations. For two TPs to communicate successfully, both must use the same type of conversation, basic or mapped. See Basic and Mapped Conversations Compared.
Type of TP: invoking or invokable	Written into the code. Invoking TPs start with TP_STARTED, which identifies the invoking TP, and ALLOCATE or MC_ALLOCATE, which identifies the requested invokable TP. Invokable TPs start with RECEIVE_ALLOCATE, which identifies the invokable TP. See Invoking TPs and Invokable TPs.
The local LU alias to be used by an invoking TP	Three options: • Written into the code in TP_STARTED. • Configured (in Host Integration Server Manager as the default local APPC LU for the user who starts the invoking TP. • Configured as a member of the default outgoing local APPC LU pool using the SNA Manager on Host Integration Server 2009. See Configuring Invoking TPs on Host Integration Server.
The invokable TP requested by an invoking TP	Written into the ALLOCATE or MC_ALLOCATE request in the invoking TP.
The LU alias to be used by an invokable TP	Two options: • Written into the invoking TP (not the invokable TP), in ALLOCATE or MC_ALLOCATE. • Configured as the default remote APPC LU for the user who starts the invoking TP. See Configuring Invoking TPs on Host Integration Server and Matching Invoking and Invokable TPs.
Type of autostarted invokable TP: queued or nonqueued	Configured with registry or environment variables. See Configuring Invokable TPs.
Local LU and remote LU aliases	Configured using SNA Manager on Host Integration Server 2009.
The pairing of local and remote LUs, and the mode used for each LU-LU pair	Configured using SNA Manager on Host Integration Server 2009.

Creating TPs and Their Supporting Configuration

The following procedure describes how to create transaction programs (TPs) and set up a supporting configuration.

To create TPs and set up a supporting configuration

1. Write, compile, and link each TP.
2. Place each TP on an appropriate computer.

   For TPs that you start many times or that are started by a user, arrange for the TP to be started easily. That is, for graphical interfaces, create a program icon for starting the TP; for non-graphical interfaces, make sure the TP is in the path.
3. On one or more servers running Host Integration Server, configure logical units (LUs), modes, and LU-LU pairs for use by the TPs.

   For information about how to set up LU-LU pairs to support TPs, see Using Invoking and Invokable TPs.
4. Set any registry or environment variables needed for the invokable TP.

   For autostarted invokable TPs, it is recommended that you use the sample TP configuration program, TPSETUP, for this step. When you write an installation program for autostarted invokable TPs, it is recommended that you include code similar to TPSETUP.

   For information about registry or environment variables, see Configuring Invokable TPs. For information about TPSETUP, see APPC Samples.
5. If the invokable TP is operator-started, start it, or arrange for it to be started when the computer is restarted and then restart the computer.

   If the invokable TP is autostarted, Host Integration Server 2009 will start it when needed.
6. Start the invoking TP.

Designing and Coding TPs

The following topics provide background information about designing and coding transaction programs (TPs).

This section contains:

- Conversation States
- Confirmation Processing
- Receiving Data Asynchronously
- Conversation Security
- Basic and Mapped Conversations Compared
- Using Invoking and Invokable TPs

Conversation States

The state of the conversation (as viewed by a particular TP) governs which APPC verbs the TP can issue at a particular time. For example, a TP cannot issue MC_SEND_DATA if the conversation is not in SEND state for that TP.

The state of a conversation depends on the TP from which it is viewed. A local TP can view a conversation as being in SEND state while the partner TP views the conversation as being in RECEIVE state. A particular TP can be in several conversations, each of which is in a different state.

The possible conversation states are summarized here.

CONFIRM
The TP has received a request for confirmation of receipt of data; it must respond positively or send error information to the partner TP.

CONFIRM_DEALLOCATE
The TP has received a request for confirmation; it must respond positively or send error information. If the TP responds positively, the conversation is automatically deallocated.

CONFIRM_SEND
The TP has received a request for confirmation; it must respond positively or send error information. After responding, the TP can begin to send data.

PENDING_POST
The TP is receiving data asynchronously. The TP can perform other processing not related to this conversation.

RECEIVE
The TP can receive application data and status information from the partner TP. When the conversation is in RECEIVE state, the TP can also send error information and request permission to send data.

RESET
The conversation has not started or has been terminated.

SEND
The TP can send data to the partner TP and request confirmation. When the conversation is in SEND state, the TP can also begin to receive data, which changes the state to RECEIVE.

SEND_PENDING
The TP issued a receive verb and the **what_rcvd** parameter returned by that verb indicated both data received and a status indication of SEND. This only affects the use of the **err_dir** parameter for SEND_ERROR and MC_SEND_ERROR. Otherwise, the state is the same as the SEND state.

This section contains:

State Checks

A state check occurs when a TP issues an APPC verb and the conversation is not in the appropriate state. For example, a state check occurs if a TP issues MC_SEND_DATA while the conversation is in RECEIVE state. When a state check occurs, APPC does not execute the verb; it returns state check information through primary and secondary return codes.

Changing Conversation States

A change in the conversation state can result from:

- A verb issued by the local TP.
- A verb issued by the partner TP.
- An error condition.

The following example shows how APPC verbs can change the state of the conversation from SEND to RECEIVE and from RECEIVE to SEND.

Note
Any TP can send or receive data, regardless of whether it is the invoking TP (the TP that started the conversation) or the invokable TP (the TP that responded to a request to start a conversation).

This example shows how APPC verbs can change the conversation state. In this table, each conversation state appears in bold and precedes the APPC verbs that are used while in that state.

Issued by the invoking TP	**Issued by the invokable TP**
TP_STARTED	
Conversation state: RESET	
MC_ALLOCATE	
(synclevel=AP_CONFIRM_SYNC_LEVL)	
Conversation state: SEND	
MC_SEND_DATA	
MC_PREPARE_TO_RECEIVE	
(ptr_type=AP_SYNC_LEVEL)	
	Conversation state: RESET
	RECEIVE_ALLOCATE
	Conversation state: RECEIVE
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)
	(what_rcvd=AP_DATA_COMPLETE)
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)

	(what_rcvd=AP_CONFIRM_SEND)
	Conversation state: CONFIRM_SEND
	MC_CONFIRMED
	Conversation state: SEND
	MC_SEND_DATA
	MC_CONFIRM
Conversation state: RECEIVE	
MC_RECEIVE_AND_WAIT	
(primary_rc=AP_OK)	
(what_rcvd=AP_DATA_COMPLETE)	
MC_RECEIVE_AND_WAIT	
(primary_rc=AP_OK)	
(what_rcvd=AP_CONFIRM_WHAT_RECEIVED)	
Conversation state: CONFIRM	
MC_REQUEST_TO_SEND	
MC_CONFIRMED	
	(rts_rcvd=AP_YES)
	MC_PREPARE_TO_RECEIVE
	(ptr_type=AP_SYNC_LEVEL)
Conversation state: RECEIVE	
MC_RECEIVE_AND_WAIT	
(primary_rc=AP_OK)	
(what_rcvd=AP_CONFIRM_SEND)	
Conversation state: CONFIRM_SEND	
MC_CONFIRMED	
Conversation state: SEND	
MC_SEND_DATA	

MC_DEALLOCATE	
(dealloc_type=AP_SYNC_LEVEL)	
	Conversation state: RECEIVE
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)
	(what_rcvd=AP_DATA_COMPLETE)
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)
	(what_rcvd=AP_CONFIRM_DEALLOCATE)
	Conversation state: CONFIRM_DEALLOCATE
	MC_CONFIRMED
Conversation state: RESET	**Conversation state: RESET**
TP_ENDED	TP_ENDED

Initial States

Before the conversation is allocated, the state is RESET for both TPs.

In the example, after the conversation is allocated, the initial state is SEND for the invoking TP and RECEIVE for the invokable TP.

Changing to RECEIVE State

MC_PREPARE_TO_RECEIVE allows a TP to change the conversation from SEND to RECEIVE state. This verb:

- Flushes the local LU's send buffer.
- Sends the AP_CONFIRM_SEND indicator to the partner TP through the **what_rcvd** parameter of a receive verb. This indicator tells the partner TP that an MC_CONFIRMED response is expected before the partner TP can begin to send data.

Confirmation processing is performed when the following conditions are true:

- The **ptr_type** parameter is set to AP_SYNC_LEVEL.
- The synchronization level of the conversation is set to AP_CONFIRM_SYNC_LEVEL.

For more information about confirmation processing, see Confirmation Processing.

Note
Issuing MC_RECEIVE_AND_WAIT while the conversation is in SEND state flushes the LU's send buffer and changes the conversation state to RECEIVE. Changing the conversation state in this manner does not support confirmation processing.

Changing to SEND State

MC_REQUEST_TO_SEND informs the partner TP (for which the conversation is in SEND state) that the local TP (for which the conversation is in RECEIVE state) wants to send data. This request is communicated to the partner TP through the **rts_rcvd**

parameter of MC_CONFIRM. (The **rts_rcvd** parameter is also returned to MC_SEND_DATA and other verbs.)

When the partner TP issues MC_PREPARE_TO_RECEIVE, the conversation state changes to RECEIVE for the partner TP, making it possible for the local TP to send data.

Note
Issuing MC_REQUEST_TO_SEND does not change the state of the conversation. Upon receiving a request to send, the partner TP is not required to change the conversation state; it can ignore the request.

Confirmation Processing

The sequence of events for confirmation processing is as follows:

1. Establish the synchronization level.
2. Send a confirmation request.
3. Receive data and confirmation request.
4. Respond to the confirmation request.
5. Deallocate the conversation.

Using confirmation processing, a TP sends a confirmation request with the data; the partner TP confirms receipt of the data or indicates that an error occurred. Each time the two TPs exchange a confirmation request and response, they are synchronized.

Note
Although the example in this section does not show this, any TP can send or receive data, regardless of whether the TP is the invoking TP or the invokable TP.

The following example illustrates confirmation processing.

Issued by the invoking TP	**Issued by the invokable TP**
TP_STARTED	
MC_ALLOCATE	
(synclevel=AP_CONFIRM_SYNC_LEVEL)	
MC_SEND_DATA	
(type=AP_SEND_DATA_CONFIRM)	
	RECEIVE_ALLOCATE
	MC_RECEIVE_AND_WAIT
MC_SEND_DATA	
(type=AP_SEND_DATA_DEALLOC_SYNC_LEVEL)	
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)
	(rtn_status=AP_YES)
	(what_rcvd= AP_DATA_COMPLETE_CONFIRM_ DEALLOCATE)
	MC_CONFIRMED

TP_ENDED	TP_ENDED

Establishing the Synchronization Level

The **synclevel** parameter of MC_ALLOCATE determines the synchronization level of the conversation. There are three possible synchronization levels:

- AP_NONE, under which confirmation processing does not occur.
- AP_CONFIRM_SYNC_LEVEL, under which the TPs can request confirmation of receipt of data and respond to requests for confirmation of data.
- AP_SYNCPT, under which the TPs operate under Sync Point Level 2 support for confirmation of receipt of data.

Sending a Confirmation Request

MC_SEND_DATA with type AP_SEND_DATA_CONFIRM has two effects:

- It flushes the local LU's send buffer and sends any data contained in the buffer to the partner TP.
- It sends a confirmation request that the partner TP receives through the **what_rcvd** parameter of a receive verb.

After issuing **MC_SEND_DATA**, the local TP waits for confirmation from the partner TP.

Receiving Data and Confirmation Request

The **what_rcvd** parameter of MC_RECEIVE_AND_WAIT indicates:

- Status of the data received: complete or incomplete.
- Future processing expected of the local TP.

In the example, **what_rcvd** is AP_DATA_COMPLETE_CONFIRM, indicating that the status is complete and a confirmation is requested.

Responding to a Confirmation Request

The partner TP issues MC_CONFIRMED to confirm receipt of data. This frees the local TP to resume processing.

Deallocating the Conversation

MC_SEND_DATA sends a confirmation request with the data when all of the following conditions are true:

- The conversation's synchronization level (established by the **synclevel** parameter of MC_ALLOCATE) is AP_CONFIRM_SYNC_LEVEL.
- The type parameter of **MC_SEND_DATA** is set to AP_SEND_DATA_DEALLOC_SYNC_LEVEL.
- The **what_rcvd** parameter of the final MC_RECEIVE_AND_WAIT is AP_DATA_COMPLETE_CONFIRM_DEALLOCATE, indicating that a confirmation of receipt of data is required before APPC will deallocate the conversation. The local TP waits for this confirmation until the partner TP issues MC_CONFIRMED.

Receiving Data Asynchronously

When using Windows 2000, a TP can receive data asynchronously, without regard to other events occurring within the TP. The following table shows the methods by which a TP can receive data asynchronously. The table also indicates how asynchronous methods can be applied to actions other than receiving data.

Operating system	Method
Windows 2000	**Through a Windows message:** Issue **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** with **WinAsyncAPPC**; the application is notified of completion through a **PostMessage** to the defined window handle. This method is not restricted to **RECEIVE_AND_WAIT** and **MC_RECEIVE_AND_WAIT**, but can be applied to any APPC verb.
Windows 2000	**Through a Win32® event:** Issue **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** with **WinAsyncAPPCEx**; the application is notified of completion through a Win32 event. This method is not restricted to **RECEIVE_AND_WAIT** and **MC_RECEIVE_AND_WAIT**, but can be applied to any APPC verb.
Windows 2000	With **RECEIVE_AND_POST** or **MC_RECEIVE_AND_POST:** Issue the **RECEIVE_AND_POST** or **MC_RECEIVE_AND_POST** verb.

The following list gives details about these methods of receiving data asynchronously. For complete information, see the verb descriptions.

RECEIVE_AND_WAITor MC_RECEIVE_AND_WAITwith WinAsyncAPPC
This method enables an application to issue a verb and be notified through a **PostMessage** when the action is complete. To retrieve the message number that will be posted to the window, call **RegisterWindowMessage** with "WinAsyncAPPC" as the input string. Then issue **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** using the **WinAsyncAPPC** entry point.

RECEIVE_AND_WAITor **MC_RECEIVE_AND_WAIT**with WinAsyncAPPCEx
This method enables an application to be notified through a Win32 event. This is particularly useful when writing applications that need to service multiple conversations simultaneously. The event must be in the nonsignaled state when passed to APPC, and the handle must have EVENT_MODIFY_STATE access to the event.

RECEIVE_AND_POSTor MC_RECEIVE_AND_POST
When using **RECEIVE_AND_POST** or **MC_RECEIVE_AND_POST** with Windows 2000, the application is notified through a Win32 event. The event must be in the nonsignaled state when passed to APPC, and the handle must have EVENT_MODIFY_STATE access to the event.

While receiving data asynchronously, the TP performs tasks not related to this conversation; the TP cannot issue most APPC verbs until notification is received. For information about the verbs that can be issued, see the descriptions of WinAsyncAPPC or WinAsyncAPPCEx.

After a verb has completed asynchronously, check the primary_rc to find out whether the data was received without error.

Note

If the initial call to issue the verb returns successfully, the application is guaranteed to be notified (by the applicable method) when the verb completes, regardless of whether the verb is ultimately successful.

Conversation Security

You can use conversation security to require that the invoking TP provide a user identifier and password before APPC will allocate a conversation with the invokable TP. If security is activated, the invoking TP must supply a combination of the user identifier and password as parameters of ALLOCATE or MC_ALLOCATE. Conversation security is activated and configured through registry or environment variables on the computer where the invokable TP is located.

With communication involving more than two TPs, the verification of a user identifier and password can be passed from one TP to another. Suppose that TP A invokes TP B, which requires security information, and TP B in turn invokes TP C, which also requires security information. Through **ALLOCATE** or **MC_ALLOCATE**, TP B can inform TP C that conversation security has already been verified.

For information about the registry or environment variables affecting conversation security, see Configuring Invokable TPs.

Basic and Mapped Conversations Compared

The following table offers some guidelines for choosing between basic and mapped conversations for your TPs. For definitions of basic and mapped conversations, see Fundamental Terms for TPs and LUs.

Characteristic	Basic conversations	Mapped conversations
Common use	Generally used for service TPs.	Generally used for application TPs.
Partnering	Must be used to communicate with an existing TP that uses basic verbs.	Must be used to communicate with an existing TP that uses mapped verbs.
Sending and receiving method	Before a TP can begin a send operation, it must convert data records into logical records. The TP does this by adding a 2-byte prefix that indicates the length of the record. A TP can send several logical records at one time. When a partner TP receives logical records, it must reconstruct them into usable data records. For more information, see Logical Records Used in Basic Conversations.	A TP sends data one record at a time. Neither the sending TP nor the receiving TP needs to convert data records between different forms.
Abnormal termination	In the DEALLOCATE verb, a TP can indicate whether an error or ABEND (abnormal program termination) was caused by a TP or by a program using the TP.	A TP can indicate an error or ABEND, but cannot tell whether a problem was caused by a TP or by a program using a TP.
	A TP can indicate whether an ABEND was caused by a timeout or by a critical error.	A TP cannot indicate the cause of an ABEND.
Error logging	For an error or ABEND, a TP can send an error message, in the form of a general data stream (GDS) error log variable, to the local log and to the partner LU.	For an error or ABEND, a TP cannot send an error message to the local log or to the partner LU.

This section contains:

- Logical Records Used in Basic Conversations
- An Example of a Mapped Conversation

Logical Records Used in Basic Conversations

Logical records are sent and received in basic conversations only.

A TP can send or receive multiple logical records with a single SEND_DATA or receive verb. The receive verbs are RECEIVE_AND_POST (Windows 2000), RECEIVE_IMMEDIATE, and RECEIVE_AND_WAIT. A TP can also send or receive a logical record in successive portions: beginning, middle, and end.

A logical record is made up of:

- A 2-byte record-length (LL) field.
- A data field that can range in length from 0 bytes through 32765 bytes.

The LL field contains a hexadecimal value that is the length of the data field plus two bytes (for the LL field). For example, if a record contains 228 bytes of application data, the logical record length is 230. The LL field is 0x00E6, the hexadecimal equivalent of 230. If the length of the data field is 0, the value contained in the LL field is 0x0002.

Logical records are sent from or received in a data buffer established by the TP. In the data buffer, the LL field must not be in Intel byte-swapped format. For example, a length of 230 must be 0x00E6, not 0xE600.

The LL field cannot be 0x0000 or 0x0001, which allow less than the two bytes required for the LL field itself. The LL field also cannot be greater than or equal to 0x8000, which is equivalent to decimal 32768 and therefore allows for a data field greater than 32765 or an LL field greater than 2.

Setting the most significant bit of the LL field to 1 indicates that the information contained in the current logical record is continued in the next logical record.

An Example of a Mapped Conversation

For background information about mapped conversations, see Basic and Mapped Conversations Compared.

The following example of a mapped conversation shows the APPC verbs used to start a conversation, exchange data, and end the conversation. APPC verb parameters are in parentheses.

Issued by the invoking TP	Issued by the invokable TP
TP_STARTED	
MC_ALLOCATE	
MC_SEND_DATA	
MC_DEALLOCATE	
TP_ENDED	RECEIVE_ALLOCATE
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_OK)
	(rtn_status=AP_NO)
	(what_rcvd=AP_DATA_COMPLETE)
	MC_RECEIVE_AND_WAIT
	(primary_rc=AP_DEALLOC_NORM)
	TP_ENDED

The following paragraphs describe the verbs that are used in a mapped conversation.

Verbs for Starting a Mapped Conversation

To start a mapped conversation, the invoking TP issues the following verbs:

- TP_STARTED, which notifies APPC that the local TP is beginning a conversation.
- MC_ALLOCATE, which requests that APPC establish a conversation between the local TP and the partner TP.

The invokable TP issues RECEIVE_ALLOCATE, which informs APPC that it is ready to begin a conversation with the invoking TP.

Verbs for Sending Data in a Mapped Conversation

MC_SEND_DATA puts one data record (a record containing application data to be transmitted) in the send buffer of the local LU. Data transmission to the partner TP does not happen until one of the following events occurs:

- The send buffer fills up.
- The sending TP issues a verb that forces APPC to flush the buffer and send data to the partner TP.

In the preceding example, the send buffer contains both the data record and the MC_ALLOCATE request (which precedes the data record). Therefore, in the example, MC_DEALLOCATE flushes the buffer, sending the **MC_ALLOCATE** request and data record to the partner TP. Other verbs that flush the buffer are MC_CONFIRM and MC_FLUSH.

Verbs for Receiving Data in a Mapped Conversation

The MC_RECEIVE_AND_WAIT verb allows a TP to receive a data record or status information. If no data is currently available, the TP waits for data to arrive. For Windows 2000 systems, issue **MC_RECEIVE_AND_WAIT** in conjunction with WinAsyncAPPC rather than the blocking version of this call.

In the example, the receiving TP issues **MC_RECEIVE_AND_WAIT** twice. The first time, it issues the verb to receive data. When it finishes receiving the complete data record (**what_rcvd** is AP_DATA_COMPLETE), it issues **MC_RECEIVE_AND_WAIT** again to receive a return code. The return code AP_DEALLOC_NORMAL indicates that the conversation has been deallocated.

Note
MC_RECEIVE_IMMEDIATE performs the same function as **MC_RECEIVE_AND_WAIT**, except that it does not wait if data is no t currently available from the partner TP. Instead, it returns a no-data-available response to the calling TP.

Verbs for Ending a Mapped Conversation

To end a mapped conversation, one of the TPs issues MC_DEALLOCATE, which causes APPC to deallocate the conversation between the two TPs.

After the conversation has been deallocated, both TPs issue TP_ENDED.

Note
A TP can participate in multiple conversations simultaneously. In this case, the TP issues **TP_ENDED** after all conversations h ave been deallocated.

Using Invoking and Invokable TPs

There are two kinds of TPs: TPs that can invoke (that is, initiate a conversation with) other TPs, and TPs that can be invoked. A TP that can invoke another TP is called an invoking TP, and a TP that can be invoked is called an invokable TP.

The topics in this section describe the following:

- How invoking TPs request invokable TPs.
- How invokable TPs identify themselves to Host Integration Server in preparation for being invoked.
- How an invokable TP is matched to an invoking TP's request.

For information about how to configure LUs to support TPs, see Configuring TPs on Host Integration Server and Host Integration Server 2009 Help.

InThis Section

Invoking TPs

An invoking TP can be located on any system on the SNA network. An invoking TP identifies itself by issuing TP_STARTED, which specifies the name of the invoking TP and can specify the LU alias that the TP uses. If the LU alias is not specified in **TP_STARTED**, Host Integration Server must be configured to supply it through one of two types of default local LU; otherwise, **TP_STARTED** will fail. For more information, see Configuring Invoking TPs on Host Integration Server.

Next, the invoking TP initiates the invoking process by issuing ALLOCATE or MC_ALLOCATE, in which it specifies the name of the invokable TP, and can also specify the partner LU alias (the LU alias to be used by the invokable TP). If the partner LU is not specified in **ALLOCATE** or **MC_ALLOCATE**, Host Integration Server 2009 must be configured to supply one through the default remote APPC LU assigned to the user who started the invoking TP; otherwise, **ALLOCATE** or **MC_ALLOCATE** will fail. For more information, see Configuring Invoking TPs on Host Integration Server.

After a TP successfully issues an **ALLOCATE** or **MC_ALLOCATE** verb, an allocation request flows. For more information about what happens after an invoking TP requests an invokable TP, see Matching Invoking and Invokable TPs.

Invoking TPs and Contention

The following information applies only to cases where LUs are communicating in complex ways (such as chains of LUs) over multiple sessions. In such cases, two LUs may attempt to allocate a conversation on the same session at the same time. If this happens, one LU must win (the contention winner) and one must lose (the contention loser). The contention-winner LU and the contention-loser LU are determined for each session when the session is established. During that particular session, the contention-loser LU must receive permission from the contention-winner LU before allocating a conversation. In contrast, the contention-winner LU on that session allocates a conversation as needed.

Note that when two LUs are communicating over multiple sessions, one LU can be the contention winner for some of the sessions, and the other LU the contention winner for others.

An invoking TP will operate most efficiently if the number of concurrent **ALLOCATE** or **MC_ALLOCATE** requests that the TP issues is matched by the number of sessions on which the local LU is the contention winner. The choice of contention winner is controlled through the modes configured at the two ends of the communication. The mode is configured using SNA Manager on Host Integration Server 2009. A mode must be configured to work with the mode on the remote system for communication to begin between two LUs. For more information about modes, see Microsoft Host Integration Server Help.

Invokable TPs

An invokable TP is a TP that can be invoked by another TP. Invokable TPs are written or configured through registry or environment variables to supply their names to Host Integration Server 2009 as a notification that they are available for incoming requests. Invokable TPs can be run on any Host Integration Server client or server running Windows 2000.

There are two types of invokable TPs:

Operator-started invokable TPs

An operator-started invokable TP must be started by an operator before the TP can be invoked. When the operator-started invokable TP is started, it notifies Host Integration Server of its availability by issuing a RECEIVE_ALLOCATE verb. The **RECEIVE_ALLOCATE** causes the name of the invokable TP, along with the alias of an associated LU if one has been configured through a registry or environment variable, to be communicated to all the servers running Host Integration Server in the SNA domain.

Autostarted invokable TPs

An autostarted invokable TP can be started by Host Integration Server when needed. The TP must be registered through registry entries or environment variables on its local system, so that it can be identified to the SnaBase component of the Host Integration Server client software. The registered information defines the TP as autostarted and must specify the TP name. The registered information can also specify the local LU alias that the invokable TP will use.

The recommended method for setting registry or environment variables for autostarted invokable TPs is to use the sample TP configuration program, TPSETUP, or similar code written into your own installation program. For more information about registry or environment variables for invokable TPs, see Configuring Invokable TPs. For information about TPSETUP, see APPC Samples.

If no local LU alias is registered with autostarted TPs, the resulting Host Integration Server configuration can be more flexible in responding to invoking requests. For more information about such flexible configurations, see TP Name Not Unique; Local LU Alias Unspecified.

After an autostarted invokable TP is started by Host Integration Server, the TP issues RECEIVE_ALLOCATE just as an operator-started TP does. **RECEIVE_ALLOCATE** must provide the TP name that was registered for the TP.

Autostarted TPs must be configured through registry or environment variables to be either queued or nonqueued. All operator-started TPs act as queued TPs.

Queued TPs

If an autostarted TP is configured as queued, or if the TP is operator-started, incoming allocation requests are queued and then sent only when the invokable TP issues **RECEIVE_ALLOCATE**. For autostarted invokable TPs, if a copy of the TP is not yet running, one is started when an incoming allocation request specifies that TP.

Note
For Windows 2000, only one copy of a service can be running at any given time; this means that all autostarted TPs that run as services under Windows 2000 must be queued. To write an autostarted TP so it will run under Windows 2000 as a service and also run in a nonqueued way, write a multithreaded program with a **RECEIVE_ALLOCATE** always outstanding.

Nonqueued TPs

If an autostarted TP is configured as nonqueued, a new copy will be started every time an ALLOCATE or MC_ALLOCATE is received for the TP. Nonqueued TPs should process the conversation they have been allocated and then exit, since they will not receive any additional **ALLOCATE** or **MC_ALLOCATE** requests.

Subcategories for Invokable TPs

The following figure shows subcategories for invokable TPs.

Queued or nonqueued	Application or service	Starting method
Queued	Running as an application or a service	Autostarted or operator-started
Nonqueued	Running as an application	Autostarted

The concept of a TP "running as a service" or "running as an application" is distinct from a service TP or an application TP. Service TP and application TP are SNA terms that describe how a TP is used: either as a supportive service program for other APPC programs, or directly by a user, as an application. For detailed information about services and applications on Windows, see the Microsoft Developer Network (MSDN®) Platform Software Development Kit.

To write an autostarted TP so it will run under Windows as a service and also run in a nonqueued way, write a multithreaded program with a RECEIVE_ALLOCATE always outstanding. See Invokable TPs.

Matching Invoking and Invokable TPs

Each computer running Host Integration Server 2009 maintains a list of available invokable TP names and any LU aliases to be associated with the TP names. This information is obtained as follows:

- For autostarted invokable TPs, registry or environment variables identify a TP name containing a maximum of eight characters, and can specify an associated LU. This information is sent from the client to the server that sponsors the client. A client learns about the domain through a sponsor connection to a server; clients must establish the sponsor connection before proceeding with any other tasks.

- For operator-started invokable TPs, a TP name (with a maximum of 64 characters) is specified with the RECEIVE_ALLOCATE verb. The TP name is truncated to eight characters and sent from the client to the server that sponsors the client, along with the alias of an associated LU if one has been configured through a registry or environment variable.

Note
If you want a TP name to be unique, it is recommended that you limit the name to eight characters or fewer, or make the name unique within the first eight characters. This is because the preliminary routing of allocation requests is carried out using the first eight characters. Although further matching is later carried out between the full TP names specified in ALLOCATE or MC_ALLOCATE and **RECEIVE_ALLOCATE**, it is inefficient to allow the preliminary routing to succeed when in some cases the later matching will fail.

The next step in the matching of invoking and invokable TPs is that the invoking TP issues the **ALLOCATE** or **MC_ALLOCATE** verb. After an invoking TP in a Host Integration Server domain successfully issues this verb, an allocation request flows to the partner LU specified in the **ALLOCATE** or **MC_ALLOCATE** verb, stating the name of the invokable TP that has been requested.

When an allocation request arrives, Host Integration Server compares the requested invokable TP name and LU alias to the list of available invokable TPs (which can include associated LU aliases). The comparison can be modified by registry variables, but by default is carried out as follows:

- Although the TP name requested in the **ALLOCATE** or **MC_ALLOCATE** verb can be as long as 64 characters, any name received through a registry or environment variable is limited to eight characters or less. Therefore, only the first eight characters of TP names are used in comparisons.

- The comparison is carried out first on both the TP name and the LU alias. An invokable TP for which there is a match on both TP name and LU alias will be chosen ahead of a TP for which no LU alias has been configured through a registry or environment variable. A TP for which no LU alias has been configured can be matched with any request that specifies that TP name, since there cannot be a mismatch based on LU alias.

- The comparison of requested and available TP names is carried out in a specific order:

    1. Host Integration Server first checks for operator-started invokable TPs on the local system (the local computer running Host Integration Server 2009).

    2. If no match is found, Host Integration Server checks for autostarted invokable TPs on the local system (the local computer running Host Integration Server 2009).

    3. If no match is found, Host Integration Server checks for operator-started invokable TPs on other Host Integration Server 2009 clients or servers.

    4. If no match is found, Host Integration Server checks for autostarted invokable TPs on other Host Integration Server clients or servers.

This comparison can be modified somewhat by registry entries for the SnaServer service. The entries are called **DloadMatchTPOnly** and **DloadMatchLocalFirst**, and are described in the *Microsoft Host Integration Server Reference online*

book.

If a match is found, Host Integration Server signals the system containing the requested TP to connect to that server running Host Integration Server 2009. If no match is found, Host Integration Server rejects the incoming request.

For suggestions about specific ways to handle TP names and LU aliases, see Arranging TPs Within an SNA Network.

Note

Because of the way APPC works, an allocation request will not flow until local data buffers are full, or a confirm or flush verb i s issued. This can mean that the allocation request does not flow until some time after the ALLOCATE or MC_ALLOCATE verb is issued. Therefore, any allocation failure caused by the rejection of the allocation request at the partner LU will be observed as the failure of a later verb with one of the allocation failure return codes.

Configuring Invokable TPs

The following topics discuss how to configure invokable transaction programs (TPs) for Microsoft Host Integration Server 2009 clients.

This section contains:

- Clients Running Windows

Clients Running Windows

On clients running Microsoft® Windows® 2000, Windows XP, and Windows Server 2003, invokable TPs are configured through the Windows registry.

Note
The recommended method for setting registry variables for autostarted invokable TPs is to use the sample TP configuration program, TPSETUP. Compile INSTALL.C, the source code for TPSETUP, for the Windows environment. When you write an installation program for autostarted invokable TPs, it is recommended that you add code similar to TPSETUP to the installation program. For information about TPSETUP, see APPC Samples.

It is recommended that autostarted invokable TPs be written as Windows services. Be sure to include code like that in TPSETUP in the program that installs your TPs. Among other things, TPSETUP shows how to use the **CreateService** function when installing a TP. For important information about how services work under Windows, see the Microsoft Developer Network (MSDN®) Platform Software Development Kit.

The following table lists the registry entries used for the types of invokable TPs that can be run on Windows clients:

Type of TP	Location in registry	Possible registry entries
Autostarted invokable TP running as a service	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services *TPName*** (and subkeys)	Registry entries created by the **CreateService** call, including entries that specify the path, display name, and other characteristics of the service. —plus— **Linkage OtherDependencies:**REG_MULTI_SZ:SnaBase **Parameters SNAServiceType:**REG_DWORD:0x5**LocalLU:**REG_SZ:*LUalias***Parameters:**REG_SZ:*ParameterList***Timeout:**REG_DWORD:*number***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }2***Username1:***REG_SZ:*Password12* ...***UsernameX:***REG_SZ:*PasswordX*2
Autostarted invokable TP running as an application	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters TPs *TPName*** Parameters	**SNAServiceType:**REG_DWORD:{ 0x5 | 0x6 }**PathName:**REG_EXPAND_SZ:*path***LocalLU:**REG_SZ:*LUalias***Parameters:**REG_SZ:*ParameterList***TimeOut:**REG_DWORD:*number***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }2***Username1:***REG_SZ:*Password12* ...***UsernameX:***REG_SZ:*PasswordX*2
Operator-started invokable TP running as a service	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services *TPName*** (and subkeys)	Registry entries created by the **CreateService** call, including entries that specify the path, display name, and other characteristics of the service. —plus— **Linkage OtherDependencies:**REG_MULTI_SZ:SnaBase **Parameters SNAServiceType:**REG_DWORD:0x1A**LocalLU:**REG_SZ:*LUalias***Timeout:**REG_DWORD:*number***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }2***Username1:***REG_SZ:*Password12* ...***UsernameX:***REG_SZ:*PasswordX*2
Operator-started invokable TP	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters TPs *TPName*** Parameters	**SNAServiceType:**REG_DWORD:0x1A**LocalLU:**REG_SZ:*LUalias***TimeOut:**REG_DWORD:*number***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }2***Username1:***REG_SZ:*Password12* ...***UsernameX:***REG_SZ:*PasswordX*2

Note
Before an autostarted TP can be started as an application on a Windows client, the TPSTART program must be started. For more information, see APPC Samples.

Note
AlreadyVerified and **Username/Password** entries are used only if **ConversationSecurity** is set to YES.

This section contains:

- Registry Entries for Clients Running Windows 2000
- Example of Windows 2000 Registry Entries for an Invokable TP

Registry Entries for Clients Running Windows 2000

The following list gives details about registry entries for clients running Windows 2000. For each TP type, the applicable variables and their locations are shown in Clients Running Windows.

Registry Entries for TPName on Clients Running Windows 2000
TPName:REG_MULTI_SZ

The name of the transaction program (TP) that is executed. A TP name is up to 64 ASCII characters in length and cannot contain spaces or nulls.

SNA service TPs are a special set of TPs defined by the SNA protocols. Each service TP is a specially-defined function with a special name. An SNA service TP name is represented by up to four EBCDIC bytes; the first byte is a hexadecimal number in the range 0x00 to 0x3F, and the remaining bytes are EBCDIC characters. The first byte defines the function class of the TP. Therefore, to convert a service TP name to an ASCII TP name form, convert the first byte as shown in the following table, and convert the EBCDIC values to ASCII letter equivalents.

First byte of TP name (hexadecimal number)	ASCII character equivalent for WIN.INI
0x07	DDM
0x20	DIA
0x21	SNAD
0x24	FS
0x30	PO
All others	UN

For example, an EBCDIC service TP name of 0x21 0xD7 0xD7 is equivalent to a TP name of SNADPP (0x21 converts to SNAD and each 0xD7 converts to P).

Registry Entries for the TPName Subtree on Clients Running Windows 2000
OtherDependencies:REG_MULTI_SZ:SnaBase

For a TP running as a service, ensures that the SnaBase service will be started before the TP is started. This entry belongs under the **Linkage** subkey.

SNAServiceType:REG_DWORD:{ 0x5 | 0x6 | 0x1A }

Indicates the type of TP. Use a value of 0x5 for an autostarted queued TP, 0x6for an autostarted nonqueued TP, and 0x1A for an operator-started TP.

Note that the value for an autostarted TP running as a service must be 0x5, because these TPs are always queued, as described in Invokable TPs.

PathName:REG_SZ **:***path*

For an autostarted TP running as an application, specifies the path and file name of the TP. The data type of REG_EXPAND_SZ means that the path can contain an expandable data string; for example, %SystemRoot% represents the directory containing the Windows 2000 system files. Note that for a TP running as a service, an equivalent entry is inserted by the **CreateService** call; no additional path entry is needed.

LocalLU:REG_SZ: *LUalias*

Specifies the alias of the local LU to be used when this TP is started on this computer.

Parameters:REG_SZ: *ParameterList*

Lists parameters to be used by the TP. Separate parameters with spaces.

Timeout:REG_DWORD: *number*

Specifies the time, in milliseconds, that a RECEIVE_ALLOCATE will wait before timing out. Specify *number* in decimal; the

registry editor converts this to hexadecimal before displaying it. The default is infinity (no limit).

ConversationSecurity:REG_SZ:{ YES | NO }

Indicates whether this TP supports conversation security. The default is NO.

AlreadyVerified:REG_SZ:{ YES | NO }

Indicates whether this TP can be invoked with a user identifier and password that have already been verified. **AlreadyVerified** is ignored if **ConversationSecurity** is set to NO. The default value is NO.

For a diagram of three TPs in a conversation, where the third TP can be invoked with a password that is already verified by the second TP, see Communication between TPs. The following table shows the requirements for using password verification in a chain of TPs.

First TP (invoking TP)	Second TP (invokable TP that confirms password and then invokes another TP)	Third and subsequent TPs (invokable TPs that invoke other TPs)
Does not need registry or environment variables.	**ConversationSecurity** setting must be YES.	**ConversationSecurity** setting must be YES.
Does not need registry or environment variables.	**AlreadyVerified** setting can be YES or NO.	**AlreadyVerified** setting must be YES.
ALLOCATE or MC_ALLOCATE in this TP has a **security** parameter of AP_PGM; as a result, the TP passes along the **user_id** and **pwd** values supplied in **ALLOCATE** or **MC_ALLOCATE**.	**ALLOCATE** or **MC_ALLOCATE** in this TP has a **security** parameter of AP_SAME; as a result, after confirming the user identifier and password, the TP passes along the user identifier and an already-verified flag.	**ALLOCATE** or **MC_ALLOCATE** in this TP has a **security** parameter of AP_SAME; as a result, the TP passes along the user identifier as received, along with the already-verified flag.

If you set **AlreadyVerified** to NO, this TP cannot join in a chain of conversations where password verification is already done. The exception to this is when **ConversationSecurity** is set to NO, in which case the TP could be the final TP in such a chain, since it performs no checking.

If you are configuring a TP that sometimes needs to confirm a password and sometimes accepts an already-verified flag, set **AlreadyVerified** to YES and configure the *UsernameX* variable appropriately. In this case, whenever the TP is invoked without the already-verified flag set, **AlreadyVerified** is ignored; verification is attempted with the user identifier and password configured for the TP.

If you want to have a chain of conversations where the user identifier and password are reverified at every step, carry out the following. For all the TPs, set **ConversationSecurity** to YES, and in each **ALLOCATE** or **MC_ALLOCATE** issued, set the security parameter to AP_PGM and the **pwd** and **user_id** parameters to valid combinations.

If you set **AlreadyVerified** to YES, make sure that **ConversationSecurity** is also set to YES.

Username1:REG_SZ:Password1 ...**UsernameX:**REG_SZ:PasswordX

Sets one or more user names and passwords to be compared with those sent by the invoking TP. The user name and password can each be as many as 10 characters. Both parameters are case-sensitive.

This variable is ignored if conversation security is not activated or if the password has already been verified, as described for the **AlreadyVerified** entry.

Example of Windows 2000 Registry Entries for an Invokable TP

For an autostarted invokable TP called **BounceTP** and running as a service, the following registry entries might be added to a Windows 2000-based client. The entries would be added to **HKEY_LOCAL_MACHINE\ SYSTEM\ CurrentControlSet\ Services\ SnaBase**, under the subkeys shown in bold type.

Note

In the following list, the parameters listed directly under the **BounceTP** key (such as **DisplayName** and **ErrorControl**) are service parameters created when TPSETUP or similar code is run to install the TP. These parameters should be created by TPSETUP or similar code; they should not be set manually. For more information about TPSETUP, see APPC Samples.

BounceTP

DisplayName:REG_SZ:BounceTP**ErrorControl:**REG_DWORD:0x1**ImagePath:**REG_EXPAND_SZ:c:\sna\system\bouncetp.exe**ObjectName:**REG_SZ:LocalSystem**Start:**REG_DWORD:0x3**Type:**REG_DWORD:0

Linkage

OtherDependencies:REG_MULTI_SZ:SnaBase

Parameters

SNAServiceType:REG_DWORD:0x5**LocalLU:**REG_SZ:JohnDoe**Parameters:**REG_SZ:Arg1 Arg2 Arg3**Timeout:**REG_DWORD:0x100**ConversationSecurity:**REG_SZ:yes**AlreadyVerified:**REG_SZ:no**JohnDoe:**REG_SZ:SecretPassword

Security

Security:REG_BINARY:

For an autostarted invokable TP called **BounceTP** running as an application, the following registry entries might be added to a Windows 2000-based client. The entries would be added to **HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\Services\SnaBase\Parameters\Tps**, under the subkeys shown in bold type.

Note

In the following list, the parameters listed under the **BounceTP** key (such as **PathName** and **ConversationSecurity**) are parameters created when TPSETUP or similar code is run to install the TP. These parameters should be created by TPSETUP or similar code; they should not be set manually. For more information about TPSETUP, see APPC Samples.

BounceTP

Parameters

SNAServiceType:REG_DWORD:0x5**PathName:**REG_SZ:C:\sna\system\bouncetp.exe**LocalLU:**REG_SZ:JohnDoe**Parameters:**REG_SZ:Arg1 Arg2 Arg3**Timeout:**REG_DWORD:0x100**ConversationSecurity:**REG_SZ:yes**AlreadyVerified:**REG_SZ:no**JohnDoe:**REG_SZ:SecretPassword

Configuring TPs on Host Integration Server

The following topics describe how configuration of invoking and invokable transaction programs (TPs) works on Host Integration Server 2009.

This section contains:

- Configuring Invoking TPs on Host Integration Server
- Configuring Invokable TPs on Host Integration Server

Configuring Invoking TPs on Host Integration Server

For a server running Host Integration Server 2009 to support the beginning of the invoking process (that is, to accept the TP_STARTED and ALLOCATE or MC_ALLOCATE verbs issued by an invoking TP), the following parameters must be configured correctly:

- If the invoking TP specifies in **TP_STARTED** the LU alias that it uses, that LU alias must match a local APPC LU alias on the supporting server running Host Integration Server 2009. If the invoking TP leaves the LU alias blank in **TP_STARTED**, one of two methods for designating a default LU must be carried out on the supporting server running Host Integration Server 2009:

    - Assign a default local APPC LU to the user or group that starts the invoking TP (that is, the user or group logged on at the system from which **TP_STARTED** is issued).

    - Designate one or more LUs as members of the default outgoing local APPC LU pool. The Host Integration Server first attempts to determine the default local APPC LU of the user who started the TP, and then attempts to assign an available LU from the default outgoing local APPC LU pool; if these attempts fail, the Host Integration Server rejects the request.

- In most situations, the supporting Host Integration Server must contain an appropriate connection to another system (host or peer). Sometimes, for testing purposes, the server running Host Integration Server 2009 contains two local LUs paired together (for invoking and invokable TPs that are in the same domain); in this situation, a connection to a host or peer is not necessary.

- If the invoking TP specifies in ALLOCATE or MC_ALLOCATE the partner LU alias, that LU alias must match an LU alias that is paired with the local LU alias specified in TP_STARTED. If the partner LU alias is left unspecified in **ALLOCATE** or **MC_ALLOCATE**, a default remote APPC LU must be assigned to the user who started the invoking TP. The default remote APPC LU is configuring using SNA Manager on Host Integration Server 2009. If the default remote APPC LU is used, it must be paired with the local LU that will be used. Otherwise, **ALLOCATE** or **MC_ALLOCATE** fails.

The preceding parameters support the beginning of the invoking process. For the invoking process to successfully complete, additional parameters must be configured as described in Configuring Invokable TPs on Host Integration Server.

Configuring Invokable TPs on Host Integration Server

For a computer running Host Integration Server 2009 to receive allocation requests from an invoking TP on another system and route those requests to an invokable TP, certain parameters must be configured correctly:

- The Host Integration Server must have a connection to the system from which the invoking TP's request is sent.

- The Host Integration Server must have a remote LU capable of receiving the incoming request. This remote LU can be configured either explicitly or implicitly.

  When configured explicitly, there is an explicit match between a remote LU alias on the Host Integration Server and the alias of the LU that conveys the invoking TP's request.

  When configured implicitly, an implicit incoming remote LU (with its implicit incoming mode) is used. This means that several items must work together. First, the LU alias specified in the incoming request (the LU alias requested for the invokable TP) must match a local LU alias on the Host Integration Server receiving the request. Second, the local LU on the Host Integration Server 2009 server must have an implicit incoming remote LU assigned to it. The properties of the implicit incoming remote LU will be used for that LU-LU session. For more details about how an implicit incoming remote LU works, see Microsoft Host Integration Server 2009 Help.

- Appropriate local LUs must be defined in the Host Integration Server configuration. For descriptions of several ways to set up these local LUs, see Arranging TPs Within an SNA Network.

Arranging TPs Within an SNA Network

If your installation of Microsoft® Host Integration Server contains multiple systems (multiple clients and/or multiple computers running Host Integration Server), you can place a given invokable transaction program (TP) on more than one system. When an invoking request is received in such an installation, there can be a choice of systems on which to run the invokable TP.

You can maintain specific control over this choice or you can allow Host Integration Server to make the choice randomly to distribute the load. To do this, follow the instructions in TP Name Not Unique; Local LU Alias Unspecified.

You can maintain specific control over the choice of system by setting up invokable TPs with unique names, or by setting up each invokable TP to run only with a specific, unique LU alias. With this arrangement, the information provided by the invoking TP (in the **ALLOCATE** or **MC_ALLOCATE** verb) specifies the system on which the invokable TP should run.

You can allow Host Integration Server to make the system choice randomly by setting the **DloadMatchLocalFirst** registry entry to NO and using invokable TPs that leave the local logical unit (LU) alias unspecified. Then, when an incoming request is received, it is routed randomly, rather than preferentially to the local Host Integration Server; in addition, no matter what LU alias is requested for the invokable TP, there cannot be a mismatch. Host Integration Server starts one instance of the requested TP, choosing randomly among the available systems.

This section contains:

- TP Name Unique for Each TP
- TP Name Not Unique; Local LU Alias Unique
- TP Name Not Unique; Local LU Alias Unspecified

TP Name Unique for Each TP

One way to specify the intended system where the invokable TP will run is to use a unique TP name for each invokable TP. In this arrangement, the invoking TP identifies the intended invokable TP (and system) simply by naming the TP. This makes it unnecessary for an invokable TP to specify any LU alias in registry or environment variables.

TP Name Not Unique; Local LU Alias Unique

Another way to specify the intended system where the invokable TP will run is to give the same name to multiple invokable TPs, but associate each TP with a unique local LU alias. To do this, configure each invokable TP (through registry or environment variables) to use a unique local LU alias. Then set up the invoking TPs so that each one is routed not only to the correct TP name but also to the correct partner LU alias for the intended invokable TP.

TP Name Not Unique; Local LU Alias Unspecified

If it does not matter on which system an invokable TP runs, use the same name for multiple invokable TPs and do not specify an LU alias in the registry or environment variables for the TPs. In this situation, there are no associated LU aliases in the list of available invokable TP names on a Host Integration Server. Thus, a request received from an invoking TP cannot cause a mismatch on the LU alias, and will match according to the TP name.

In this situation, if you set the **DloadMatchLocalFirst** registry entry to NO, Host Integration Server randomly routes the request to one of the available TPs. This spreads the processing load among multiple systems and provides hot backup (the ability to take systems online and offline without disrupting service).

Troubleshooting for Invokable TPs

If there are difficulties with starting an invokable TP, there may be a mismatch between the information for the invokable TP, the invoking TP, and/or LUs in the Host Integration Server 2009 configuration. That is, there may be a mismatch between the parameters for RECEIVE_ALLOCATE, TP_STARTED, ALLOCATE, or MC_ALLOCATE and/or LU aliases specified in server configuration. LU aliases are configured using SNA Manager on Host Integration Server 2009.

Simplifying APPC Configuration

There are several features in Host Integration Server 2009 that can simplify configuration for APPC:

- The implicit incoming remote LU and the implicit incoming mode, which allow Host Integration Server to accept requests that arrive by unrecognized remote LUs and modes.
- The default local APPC LU and the default remote APPC LU, which allow LU aliases to be associated with user or group names, simplifying the routing of incoming requests and the configuration of client systems.
- The default outgoing local APPC LU pool, which allows LUs to be allocated dynamically to any invoking TP that does not specify a local LU.
- Automatic partnering, which automatically creates LU-LU pairs and assigns modes to the pairs.

Sync Point Level 2 Support in Host Integration Server

The following topics describe additions to Host Integration Server 2009 that enable Sync Point Level 2 support to the LU 6.2 protocol stack.

This section assumes familiarity with the existing Host Integration Server APPC basic and mapped conversation interfaces. It does not attempt to explain the SNA formats and protocols for implementing Sync Point protocols, which are described in *SNA LU6.2 Reference: Peer Protocols* published by IBM (Document SC31-6808-1).

This section contains:

Sync Point Functional Overview

The Sync Point support additions to Host Integration Server allow vendors to provide Sync Point services over LU 6.2 conversations provided by the LU 6.2 protocol stack in Host Integration Server. These additions do not implement the architected Sync Point components and TPs necessary for a complete Sync Point implementation. In particular, the following Sync Point components are not implemented, and must be provided by the vendor.

- Sync Point Services (SPS)
- Conversation-Protected Resource Manager (C-PRM)
- Resynchronization TP

These vendor-supplied components and applications are expected to implement the **SYNCPT** and **BACKOUT** verbs used for Sync Point services. The **SYNCPT** verb is used to synchronize transactions. The **BACKOUT** verb is used to back out of a transaction.

SPS, C-PRM, and the Resynchronization TP are specific components of the SNA Sync Point architecture described in *SNA LU6.2 Reference: Peer Protocols* published by IBM.

Host Integration Server has been modified to add the features necessary to support these components, namely:

- Additions to the existing APPC API to support implementation of Sync Point verbs.
- Accounting support for Sync Point protocols.
- Modifications to invokable transaction program (TP) initiation.

Changes to the APPC basic and mapped conversation APPC APIs are made so as to ensure backward compatibility with existing APPC applications that adhere strictly to the API.

Note
Applications must zero all reserved verb control block (VCB) members before issuing an APPC verb. If this is not done, the application may inadvertently invoke one of the new APPC features.

Sync Point Support Architecture

The Sync Point support provided by Host Integration Server 2009 assumes a particular implementation architecture by the vendor, as follows:

- The vendor provides a communication interface to its own clients requiring Sync Point Services (SPS).
- The vendor API maps its communication and Sync Point requests to the Host Integration Server APPC API.
- The vendor provides a single Microsoft Windows 2000 process, the Transaction Monitor, that is responsible for:
    - Issuing all APPC verbs.
    - Implementing the architected Resynchronization TP.
    - Implementing the architected Conversation-Protected Resource Manager (C-PRM) component of the logical unit (LU).
    - Implementing the architected SPS component of the LU.

  The Transaction Monitor must reside on the same computer as the Host Integration Server containing the LUs for which it is providing Sync Point services. Both incoming and outgoing Sync Point conversations for this Transaction Monitor will be routed through this Host Integration Server 2009 server only.

Detailed descriptions of the three architected Sync Point components can be found in *SNA LU6.2 Reference: Peer Protocols* published by IBM.

Sync Point Session Support

This section discusses support for Sync Point session activation and deactivation in Host Integration Server 2009.

In This Section

Sync Point Session Activation

If Host Integration Server 2009 is to support Sync Point conversations, this must be specified at session activation time. The configuration of Host Integration Server is modified to allow the system administrator to specify which (if any) local LUs will be used for Sync Point conversations.

The **Local LU Configuration** property page in Host Integration Server contains a new check box. When checked, it indicates that the local LU can participate in Sync Point sessions. Host Integration Server uses this option to determine the parameters it sends on BIND requests and responses.

When Host Integration Server initiates an LU 6.2 session on an LU designated as supporting Sync Point, it sets the synchronization level on BIND requests to indicate that the session can support Sync Point and Backout. If the partner LU also supports Sync Point and Backout, the session is available for conversations requiring Sync Point support. If the partner LU does not support Sync Point, the session will not be used for Sync Point conversations.

Similarly, if the local LU is configured for Sync Point and the partner LU's BIND request indicates that it supports Sync Point, Host Integration Server sends BIND responses specifying that Sync Point is supported. In this case, the session can be used for Sync Point conversations.

Sync Point Session Deactivation

A Sync Point implementation needs to determine whether it has lost connectivity to a partner when establishing Sync Point conversations so that it can know whether or not to resynchronize. To obtain this information, Host Integration Server provides a new APPC verb, GET_LU_STATUS that reports the status of a particular remote LU. The information returned by this verb is as follows:

- Current number of active LU 6.2 sessions between the remote LU and the TP's local LU.
- Whether or not the number of active sessions dropped to zero at any time since this verb was last issued for the remote LU.

Note that the zero sessions indicator is reset to AP_NO each time the verb is issued by any process. It is therefore imperative that only one process issues this verb; otherwise information may be lost.

Starting Local Sync Point TPs

Local TPs are created by issuing the TP_STARTED verb to Host Integration Server. The **TP_STARTED** verb has been modified by adding the new verb control block (VCB) member **syncpoint_rqd** to allow a TP to specify that it requires Sync Point services.

By setting **syncpoint_rqd** to AP_YES, a TP indicates that it requires Sync Point services from Host Integration Server. A value of AP_NO (the default) indicates that Sync Point services are not required.

Since this member cannot be incorporated within the existing **TP_STARTED** VCB, the TP must use a larger VCB structure. To indicate that the VCB is longer than usual, the **opext** member of the VCB must be combined using OR with the value AP_EXTD_VCB before calling APPC.

Conversations started by TPs requiring Sync Point support will be routed only by the Host Integration Server software running on the same computer. They will not be routed to other LAN-attached servers.

Sync Point Conversation Activation

This section discusses support for Sync Point conversation activation in Host Integration Server.

This section contains:

- Locally Initiated Conversations
- Remotely Initiated Conversations
- Already Verified Support
- Presentation Header Support in Data Transfers
- User Control Data
- Implied Forget

Locally Initiated Conversations

Conversations are initiated locally by issuing an ALLOCATE or MC_ALLOCATE verb. The **ALLOCATE** and **MC_ALLOCATE** verbs are modified to support additional parameters required by Sync Point support. The supplied **synclevel** parameter of the **ALLOCATE** and **MC_ALLOCATE** verbs can take on a value of AP_SYNCPT, which specifies that the conversation requested is a Sync Point conversation.

Remotely Initiated Conversations

Applications that want to receive remotely initiated conversations (incoming Attaches) issue a RECEIVE_ALLOCATE verb. To accommodate Sync Point support, the **RECEIVE_ALLOCATE** verb is modified in a number of ways as follows:

- The returned **sync_level** parameter of the **RECEIVE_ALLOCATE** verb can take on a value of AP_SYNCPT, specifying that the conversation is a Sync Point conversation. The value of the **sync_level** parameter can also be determined by issuing a GET_ATTRIBUTES verb on the new conversation.

- Support is added for the receipt of program initiation parameters (PIP) data by a new parameter to the **RECEIVE_ALLOCATE** verb:

  The **pip_incoming** parameter is set by the application to indicate whether it is willing to accept incoming PIP data, and is returned by Host Integration Server to indicate whether PIP data is available to be received. If the application does not want to receive PIP data, this member should be set to AP_NO, the default, before issuing the **RECEIVE_ALLOCATE** verb. If it is willing to accept PIP data, this member should be set to AP_YES. On completion of the **RECEIVE_ALLOCATE** verb, this member will be set to AP_YES if PIP data is available to be received by the application and to AP_NO otherwise.

- If PIP data is available, the application can receive it by issuing one of the verbs for receiving data on completion of the RECEIVE_ALLOCATE verb. For basic conversations, these receive verbs include RECEIVE_AND_POST, RECEIVE_AND_WAIT, and RECEIVE_IMMEDIATE. On basic conversations the PIP data will be returned inclusive of the general data stream (GDS) header for PIP data (GDS identifier 0x12F5). For mapped conversations, these receive verbs include MC_RECEIVE_AND_POST, MC_RECEIVE_AND_WAIT, and MC_RECEIVE_IMMEDIATE. On mapped conversations, Host Integration Server 2009 removes the 4-byte GDS header, and returns the PIP data only.

- For basic conversations, if the application issues a SEND_ERROR, DEALLOCATE, or TP_ENDED verb before the PIP data is received, the PIP data will be discarded. For mapped conversations, if the application issues an MC_SEND_ERROR, MC_DEALLOCATE, or **TP_ENDED** verb before the PIP data is received, the PIP data will be discarded.

- If PIP data is received for a TP that cannot or does not want to receive it, the conversation is rejected with a primary return code of AP_ALLOCATION_ERROR, and a secondary return code of AP_PIP_NOT_ALLOWED.

Already Verified Support

In an implementation where a Host Integration Server application acts as a gateway between an SNA network and a non-SNA network, it is possible that non-Host Integration Server clients of the gateway may require Sync Point Level 2 conversation security. Since the originating client will have validated the relevant user identifier and password, the gateway application should specify conversation security of AP_SAME when starting a conversation on behalf of the client. In this case, however, Host Integration Server assumes that the user identifier to be used has previously been received on an Attach targeted at the TP. In the case of a non-Host Integration Server client this is not the case.

To allow such a gateway to support Sync Point Level 2 conversation security, Host Integration Server provides a new verb, SET_TP_PROPERTIES, which allows the gateway application to set the user identifier for the TP before allocating a conversation with security of AP_SAME. This verb will normally be issued once, immediately after TP_STARTED, to set the user identifier for all the TP's conversations.

Presentation Header Support in Data Transfers

For basic conversations, Sync Point commands are sent by means of presentation headers (PS) across LU 6.2 conversations using the SEND_DATA or MC_SEND_DATA verb. All presentation headers contain length fields that specify a length of 1, which is usually illegal. To support Sync Point conversations, the following modifications are made to the Host Integration Server presentation services component:

- On basic conversations with a **synclevel** of AP_SYNCPT, data transferred specifying a general data stream (GDS) variable length of 1 will not be rejected. If the **synclevel** is not AP_SYNCPT, they will be rejected as before.

- On mapped conversations, PS headers will not be wrapped as mapped conversation application data logical records (with GDS identifier 0x12FF) when they are sent, or have the GDS header stripped off when they are received.

- On mapped conversations, it is the responsibility of the application to provide the complete PS header including the length field. Similarly, the length field will be included in PS header data returned by receive verbs.

To achieve the latter the MC_SEND_DATA verb and the receive verbs (MC_RECEIVE_AND_POST, MC_RECEIVE_AND_WAIT, and MC_RECEIVE_IMMEDIATE) require modifications as follows:

- A new parameter, **data_type**, is added to the **MC_SEND_DATA** verb. When this is set to AP_APPLICATION (the default, 0x00), the data is sent as application data (GDS identifier 0x12FF) as usual. When it is set to AP_PS_HEADER, the data is sent as described above.

- The following two new values are added for the **what_rcvd** member of the receive verbs to specify that the received data is a PS header:

  AP_PS_HEADER_COMPLETE

  AP_PS_HEADER_INCOMPLETE

- If an application issues a receive verb with **rtn_status** set to AP_YES, Host Integration Server will return status in combination with AP_PS_HEADER_COMPLETE, with the exception of AP_DEALLOCATE_NORMAL and AP_CONFIRM_DEALLOCATE. This is to prevent the conversation being prematurely disconnected from the LU 6.2 session when a COMMIT PS header arrives with the end of conversation indication.

It is the responsibility of the vendor-supplied Sync Point support component to convert these PS headers into the appropriate Sync Point return codes (for example, TAKE_SYNCPT).

User Control Data

For mapped conversations, the MC_SEND_DATA verb and the receive verbs (MC_RECEIVE_AND_POST, MC_RECEIVE_AND_WAIT, and MC_RECEIVE_IMMEDIATE) are modified to allow applications to send and receive data in user control data general data stream (GDS) variables instead of the regular application data GDS variables. The **MC_SEND_DATA** verb is modified as follows:

- A new parameter, **data_type**, is added. When **data_type** is set to AP_USER_CONTROL_DATA, the data is sent as user control data (GDS identifier 0x12F2). When it is set to AP_APPLICATION (the default), the data is sent as application data (GDS identifier 0x12FF). Note that the APPC library automatically creates the GDS header on behalf of the application for both AP_APPLICATION and AP_USER_CONTROL_DATA data records.

- The mapped conversation receive verbs are modified to allow applications to receive user control data by adding two new values for the **what_rcvd** parameter, as follows:

  AP_USER_CONTROL_DATA_COMPLETE

  AP_USER_CONTROL_DATA_INCOMPLETE

Implied Forget

LU 6.2 Sync Point sessions can use an optimization of the architected message flows known as implied forget. When the protocol specifies that a FORGET presentation header (PS) is required, the next data flow on the session implies that a FORGET has been received, even though it has not. In the normal situation, the TP is aware of the next data flow when data is received or sent on one of its Sync Point conversations.

However it is possible that the last message that flows is caused by the conversation being deallocated. In this case, the TP is unaware when the next data flow on the session occurs. To provide the TP with this notification, the DEALLOCATE and MC_DEALLOCATE verbs are modified to allow the TP to register a callback function which will be called:

- On the first normal flow transmission (request or response) over the session used by the conversation.
- If the session is unbound before any other data flows.
- If the session is terminated abnormally due to a data link control (DLC) outage.

The callback procedure can take any name because the address of the procedure is passed into the APPC DLL.

Note that the **DEALLOCATE** and **MC_DEALLOCATE** verbs will probably complete before the callback routine is called. The conversation is considered to be in RESET state and no further verbs can be issued using the conversation identifier. If the application issues a TP_ENDED verb before the next data flow on the session, the callback routine will not be invoked.

The DEALLOCATE and MC_DEALLOCATE verbs are modified as follows to support implied forget:

- A new member, **callback**, is added to allow the TP to specify the address of the function to call on the next data flow on the session being used by the conversation being deallocated. If this member is NULL, no notification will be provided. A vendor would normally supply this callback function.
- The **DEALLOCATE** and **MC_DEALLOCATE** verbs also contain a correlator member which is returned as one of the parameters when the callback function is invoked. The application can use this parameter in any way (for example, as a pointer to a control block within the application).

Host Integration Server allows TPs to deallocate conversations immediately after sending data by specifying the **type** member in the SEND_DATAand MC_SEND_DATAverbs as AP_SEND_DATA_DEALLOC_FLUSH, AP_SEND_DATA_DEALLOC_SYNC_LEVEL, AP_SEND_DATA_DEALLOC_ABEND, and AP_SEND_DATA_DEALLOC_CONFIRM. However, the **SEND_DATA** and **MC_SEND_DATA** verbs do not contain the implied forget callback function. TPs wishing to receive implied forget notification must issue a DEALLOCATE or MC_DEALLOCATE verb explicitly.

Sync Point Level 2 Confirm Support

The current APPC implementation in Host Integration Server supports conversations with **synclevel** of AP_NONE, AP_CONFIRM_SYNC_LEVEL, or AP_SYNCPT. The DEALLOCATE, MC_DEALLOCATE, PREPARE_TO_RECEIVE, and MC_PREPARE_TO_RECEIVE verbs specify a **type** member indicating the synchronization level required. This parameter is interpreted as follows:

Allocated synclevel	Type specified	Action performed
AP_NONE	AP_FLUSH	Action of FLUSH or MC_FLUSH verb before deallocation or change of direction.
AP_NONE	AP_SYNCLEVEL	Action of **FLUSH** or **MC_FLUSH** verb before deallocation or change of direction.
AP_SYNCPT	AP_FLUSH	Action of FLUSH or MC_FLUSH verb before deallocation or change of direction.
AP_SYNCPT or AP_CONFIRM_SYNC_LEVEL	AP_CONFIRM_TYPE	Action of CONFIRM or MC_CONFIRM verb before deallocation or change of direction.
AP_SYNCPT	AP_SYNCLEVEL	It is assumed that a Sync Point implementation built using the APPC API in Host Integration Server implements the defer states appropriately. See the note below.

Note

With an allocated **synclevel** of AP_SYNCPT and a specified **type** of AP_SYNCLEVEL, it is assumed that a vendor-supplied Sync Point component implements the defer states appropriately. A vendor-supplied Sync Point system must:

- Intercept DEALLOCATE, MC_DEALLOCATE, PREPARE_TO_RECEIVE, and MC_PREPARE_TO_RECEIVE verbs on Sync Point Level 2 conversations when type AP_SYNCLEVEL is specified for **synclevel**.
- Maintain the defer state until one of the verbs valid in that state completes.
- On completion of the verb, issue the original **DEALLOCATE**, **MC_DEALLOCATE**, **PREPARE_TO_RECEIVE**, or **MC_PREPARE_TO_RECEIVE** verb to Host Integration Server 2009.

Host Integration Server does not implement the defer states directly. In particular, when a **DEALLOCATE**, **MC_DEALLOCATE**, **PREPARE_TO_RECEIVE**, or **MC_PREPARE_TO_RECEIVE** verb is received with a **type** specified as AP_SYNCLEVEL on a Sync Point conversation, this is treated as if the conversation has a **synclevel** of AP_NONE.

So that Sync Point Level 2 conversations can use confirm type synchronization, the **DEALLOCATE**, **MC_DEALLOCATE**, **PREPARE_TO_RECEIVE**, and **MC_PREPARE_TO_RECEIVE** verbs are modified to support a type member of AP_CONFIRM_TYPE.

The DEALLOCATE, MC_DEALLOCATE, PREPARE_TO_RECEIVE, and MC_PREPARE_TO_RECEIVE verbs specify a type member indicating the synchronization level required. This parameter is interpreted as follows:

Allocated synclevel	Type specified	Action performed
AP_NONE	AP_FLUSH	Action of FLUSH or MC_FLUSH verb before deallocation or change of direction.
AP_NONE	AP_SYNCLEVEL	Action of **FLUSH** or **MC_FLUSH** verb before deallocation or change of direction.
AP_CONFIRM_SYNC_LEVEL	AP_FLUSH	Action of **FLUSH** or **MC_FLUSH** verb before deallocation or change of direction.
AP_CONFIRM_SYNC_LEVEL	AP_SYNCLEVEL	Action of CONFIRM or MC_CONFIRM verb before deallocation or change of direction.

Sync Point Backout Support

This section describes back out support for Sync Point conversations.

This section contains:

Additional Sync Point Return Codes

When a remote transaction program (TP) issues a **BACKOUT** verb, the back out is reported to the local TP as a new primary return code value, AP_BACKED_OUT, on the next (current) verb issued. The local TP is provided access to the sense code information contained in the Backout FMH-7 by setting the **secondary_rc** field as follows:

- AP_BO_NO_RESYNC for sense code 0x08240000
- AP_BO_RESYNC for sense code 0x08240001

This new return code will only be supplied on conversations with **synclevel** AP_SYNCPT, and therefore will not be presented to existing applications.

The verbs on which this new return code can be returned are:

CONFIRM

MC_CONFIRM

MC_PREPARE_TO_RECEIVE

MC_RECEIVE_AND_POST

MC_RECEIVE_AND_WAIT

MC_RECEIVE_IMMEDIATE

MC_SEND_DATA

MC_SEND_ERROR

PREPARE_TO_RECEIVE

RECEIVE_AND_POST

RECEIVE_AND_WAIT

RECEIVE_IMMEDIATE

SEND_DATA

SEND_ERROR

Sending Backout on Sync Point Conversations

To send a Backout, an FMH-7 containing a sense code of 0x08240000 or 0x08240001 is sent on the session. This is done using the SEND_ERROR or MC_SEND_ERROR verb. To enable Host Integration Server 2009 to send the appropriate sense data, the **SEND_ERROR** and **MC_SEND_ERROR** verbs are modified as follows:

- A new field, **err_type**, is added to allow the TP to specify the type of error. The default is AP_PROG (0x00), which means existing TPs will continue to work unmodified.

- The **err_type** field in both verbs can take one of two new values, specifying the sense codes to be generated by Host Integration Server 2009:

  AP_BACKOUT_NO_RESYNC for sense code 0x08240000

  AP_BACKOUT_RESYNC for sense code 0x08240001

LUWID, Conversation Correlators, and Session Identifiers

The logical unit-of-work identifier (LUWID), conversation correlators, and session identifiers are important for all Sync Point operations and accounting purposes. The following sections describe how Host Integration Server provides access to these components and, where appropriate, facilities to modify this information.

This section contains:

Generating and Setting LUWIDs

The unit-of-work identifier (LUWID) is used to identify conversations that are part of a single Sync Point transaction. All conversations with the same LUWID are committed (or backed out) at the same time.

Host Integration Server assigns two LUWIDs to a transaction program when the TP is started. For locally started TPs, this is when the TP_STARTED verb is issued. The first LUWID is the TP's protected LUWID. It is used by Host Integration Server 2009 as the LUWID for all **synclevel** AP_SYNCPT conversations allocated by the TP. When the TP issues an ALLOCATE or MC_ALLOCATE verb with a **synclevel** of AP_SYNCPT, Host Integration Server generates an Attach containing the TP's current protected LUWID.

The second LUWID is the TP's unprotected LUWID. It is used on all conversations allocated by the TP with a **synclevel** other than AP_SYNCPT.

For remotely initiated TPs, the incoming Attach may contain an LUWID for the TPit is mandatory if the conversation has a **synclevel** of AP_SYNCPT. For Sync Point conversations, Host Integration Server saves the LUWID as the TP's protected LUWID and generates a new unprotected LUWID for it. For conversations with a **synclevel** other than Sync Point (AP_SYNCPT), Host Integration Server saves the LUWID as the TP's unprotected LUWID and generates a new protected LUWID.

Host Integration Server generates LUWIDs by concatenating the following:

- The fully qualified name of the local LU, preceded by a single byte indicating its length (exclusive of the length byte).
- A 6-byte LUW instance number, generated from the current date and time (modified to ensure uniqueness if necessary).
- A 2-byte LUW sequence number, initialized to 1.

If the fully qualified LU name component of the LUWID is not 17 bytes long, Host Integration Server does not add any padding between it and the LUW instance number. The application can determine the length of the LUWID, and the offsets within it of the LUW instance number and LUW sequence number, by examining the first byte of the LUWID, which indicates the length of the fully qualified LU name.

When Host Integration Server generates both a protected and an unprotected LUWID for a TP, the unprotected LUWID is created by incrementing the protected LUWID's instance number.

The protected LUWID needs to be changed by a TP for one of four reasons:

- When a transaction is backed out or committed, the LUWID sequence number must be incremented.
- If the transaction tree is split, a new LUWID must be generated for the TP.
- If the application uses multiple logical TPs to implement a transaction, each TP must have the same LUWID (different from that assigned by Host Integration Server).
- If the application is acting as a gateway from a non-SNA environment and LUWIDs are received by a means other than an Attach.

To allow a TP to set or generate new LUWIDs, a new verb, SET_TP_PROPERTIES, is provided by the APPC API. This verb allows the TP to either set its LUWIDs to an existing value, by providing the LUWIDs, or generate new ones and use them from then on. When a new LUWID is generated by Host Integration Server, it is guaranteed to be unique.

Note that it is the responsibility of the application (the Sync Point system component) to transmit the new LUWID PS header to the partner Sync Point system when the protected LUWID is changed. Similarly, when a new LUWID PS header is received, the application must inform the LU by issuing **SET_TP_PROPERTIES**.

Extracting LUWIDs

Both LUWIDs for a particular TP can be determined by issuing the GET_TP_PROPERTIES verb. The **GET_TP_PROPERTIES** verb returns the TP's unprotected LUWID in the **luw_id** field.

If the TP needs to access the protected LUWID, it must combine the **opext** member of the verb control block (VCB) with the value AP_EXTD_VCB using OR before issuing the verb. The protected LUWID will then be returned in the **prot_luw_id** field. If the **opext** field does not contain the AP_EXTD_VCB bit, the verb control block is presumed to end immediately before the **prot_luw_id** field.

The LUWID for a particular conversation can be determined by issuing a GET_ATTRIBUTES or MC_GET_ATTRIBUTES verb on the conversation. These verbs are modified as follows:

- A new field, **luw_id**, is added in which the LUWID is returned. The LUWID returned is the protected one if the conversation was allocated with **synclevel** field of the ALLOCATE or MC_ALLOCATE verb set to Sync Point (AP_SYNCPT); otherwise it is the unprotected one.

- Since the **luw_id** field cannot be incorporated within the existing verb control blocks, the TP must use a larger VCB structure. To indicate that the VCB is longer than usual, the **opext** field of the VCB must be combined with the value AP_EXTD_VCB using OR before calling APPC.

- The **sync_level** field of the **GET_ATTRIBUTES** or **MC_GET_ATTRIBUTES** verb can take an additional value, AP_SYNCPT, when the conversation was allocated with the **synclevel** field of the **ALLOCATE** or **MC_ALLOCATE** verb of Sync Point (AP_SYNCPT).

Session Identifiers

Host Integration Server maintains a unique identification for every LU 6.2 session it has with a remote LU. This 8-byte identifier is generated by Host Integration Server every time it starts a new session (or is received by Host Integration Server when a session is initiated remotely). The Sync Point resynchronization protocols require knowledge of the session identifier.

To provide this, the MC_GET_ATTRIBUTES and the GET_ATTRIBUTES verbs have been modified to return the session identifier of the session over which a particular conversation is allocated. The **MC_GET_ATTRIBUTES** and **GET_ATTRIBUTES** verbs can be used to retrieve this **sess_id** field of the VCB if the **opext** field of the VCB is combined with the value AP_EXTD_VCB using OR before calling APPC.

Configuration Changes for Sync Point Support

A new check box is added to the Local LU Configuration dialog box. When selected, this indicates that the local LU is able to participate in **synclevel** Sync Point sessions. Host Integration Server uses this option to determine the **synclevel** BIND parameters it sends on BIND requests and responses.

This field is added to the Host Integration Server configuration file in a field that is no longer used by Host Integration Server. Existing configurations from earlier versions of Host Integration Server will therefore continue to work unmodified.

Accepting Incoming Attaches

The Sync Point support in Host Integration Server is intended for use only by gateway applications that implement the architected SNA Sync Point components, including Conversation-Protected Resource Manager (C-PRM). In a Sync Point implementation, it is necessary for C-PRM to be aware of all protected conversations, both locally initiated and remotely initiated. This can be achieved in Host Integration Server by C-PRM intercepting the conversation allocation and deallocation verbs and issuing them on behalf of the transaction program (TP). Note that since Host Integration Server does not allow TP or conversation identifiers to be shared across processes, this also means that the process containing C-PRM must also intercept all APPC verbs issued by the client TPs.

For locally initiated TPs, this is straightforward. However for incoming Attaches, the situation is made more complex by the requirement that the RECEIVE_ALLOCATE verb specify the name of the TP to be matched with the Attach.

In some implementations, this will not be an issue, as the gateway will be aware of the names of all the transactions passing through it. To support this situation, the **RECEIVE_ALLOCATE** verb has been enhanced as described in the following topic to permit the gateway to indicate that it can accept Sync Point conversations.

In other implementations, the gateway does not know the names of the transactions passing through it. This is particularly so when the gateway is providing a conversion between SNA and another communications protocol. In this case, Host Integration Server allows the gateway process to register itself as a Sync Point Attach Service, indicating that it is willing to accept incoming Attaches for any Sync Point conversation. In this case, the gateway must be implemented as a Sync Point Attach Manager.

This section contains:

- Sync Point Knows Transaction Names
- Sync Point Attach Manager
- Rejecting Remotely Initiated Conversations

Sync Point Knows Transaction Names

A Sync Point implementation that knows the names of all the transactions that can be supported (for example, through configuration of the gateway) may accept incoming Sync Point conversations by issuing a RECEIVE_ALLOCATE verb specifying the name of the transaction and indicating that it is willing to accept Sync Point conversations.

The **RECEIVE_ALLOCATE** verb was modified to allow a TP to specify that it can accept Sync Point conversations by adding a new **syncpoint_rqd** field to the VCB. When this field is set to AP_YES it indicates that the transaction program can accept Sync Point conversations from Host Integration Server. When this field is set to AP_NO (the default), it indicates that Sync Point conversations are not supported.

Sync Point Attach Manager

Instead of issuing separate RECEIVE_ALLOCATE verbs for each possible transaction name, a Sync Point implementation may instead register as the Sync Point Attach Manager for Host Integration Server 2009. It does so by issuing a **RECEIVE_ALLOCATE** verb specifying a TP name consisting of all 0x00s.

When a Sync Point Attach Manager is registered, the following changes are effected in server's incoming Attach support on Host Integration Server:

- When an Attach message arrives for any TP name on a conversation with the **syncpoint_rqd** field of the VCB set to AP_YES, Host Integration Server matches it with the application that issued the special **RECEIVE_ALLOCATE** verb registering itself as the Sync Point Attach Manager.

- Any Attach message arriving for the Resynchronization TP (0x06F2) will automatically be routed to the Sync Point Attach Manager.

- If no **RECEIVE_ALLOCATE** has been issued for the Sync Point Attach Manager, or for the specific TP name, Host Integration Server will queue the Attach for a configured period of time. If no **RECEIVE_ALLOCATE** is issued in that time, the Attach will be rejected with a return code of TP_NOT_AVAILABLE_RETRY.

- If a **RECEIVE_ALLOCATE** is matched with the Attach message, the verb is returned to the TP with the **tp_name** field of the VCB set to the TP name contained in the Attach message.

Applications using this feature must adhere to two restrictions:

- All verbs issued on conversations started in this manner must be issued by the same process, as Host Integration Server cannot pass **tp_id**s between processes.

- Only a single process may register as the Sync Point Attach Manager on any server running Host Integration Server. If a second process attempts to register, its **RECEIVE_ALLOCATE** verb will return immediately with the primary return code set to AP_SYNCPOINT_MANAGER_ACTIVE.

Sync Point Attach Manager applications must reside on a Host Integration Server 2009 server. They may not be distributed across Host Integration Server clients. This restriction is imposed to ensure that only a single instance of Sync Point Services (SPS) and Conversation-Protected Resource Manager (C-PRM) exists for each LU on the Host Integration Server (which might not be the case if Sync Point Attach Managers were visible from multiple servers in the Host Integration Server domain).

The structure of the RECEIVE_ALLOCATE verb control block does not require modification to support this function.

Rejecting Remotely Initiated Conversations

In an environment where a Sync Point Attach Manager is receiving all Attach messages as described above, it may be necessary for it to reject an Attach for a particular TP name, either because the TP name is not valid or because there is another problem with the received Attach message. To enable the application to generate the correct return code at the initiating TP, the DEALLOCATE and MC_DEALLOCATEverbs are enhanced with new **deallocate_type** field values in the VCB that allow the application to specify the return code to be sent to the initiating TP. The new values for **deallocate_type** are:

AP_TP_NOT_AVAIL_RETRY

AP_TP_NOT_AVAIL_NO_RETRY

AP_TPN_NOT_RECOGNIZED

AP_PIP_DATA_NOT_ALLOWED

AP_PIP_DATA_INCORRECT

Sync Point Examples

This section contains example verb sequences for implementing the architected Sync Point verbs using the Sync Point facilities provided by Host Integration Server.

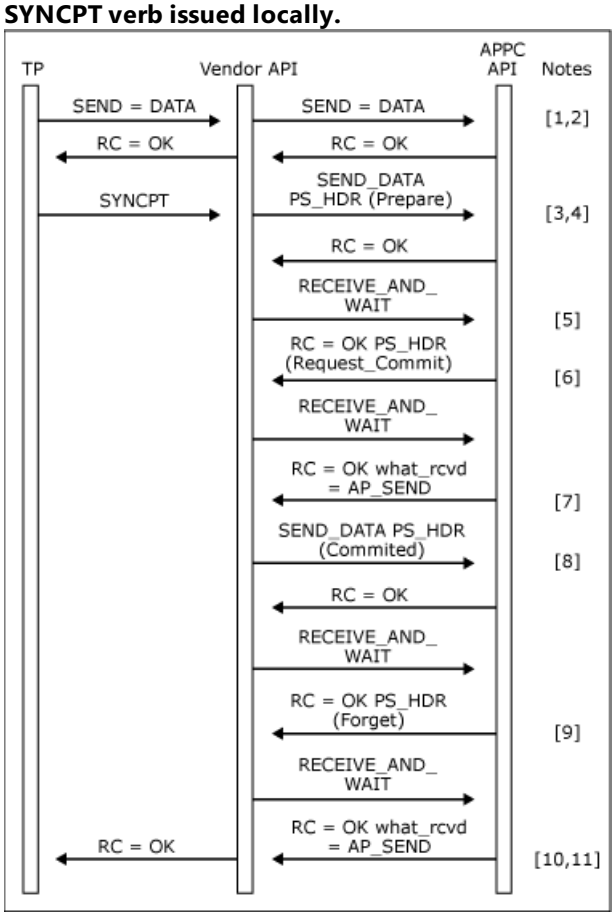
In the following figures, TP is the transaction program that requires Sync Point services. Vendor API is the vendor-supplied APPC API. This component provides the SPS and C-PRM components and a mapping between the vendor's APPC syntax and that of Host Integration Server. APPC API is the Host Integration Server APPC basic and mapped conversation interface.

This section contains:

SYNCPT Verb Issued Locally

This section provides an example verb sequence with a verb issued locally.

SYNCPT verb issued locally.



1. The transaction program issues a SEND_DATA or MC_SEND_DATAverb depending on whether a basic or mapped conversation is being used.

2. The **SEND_DATA** or **MC_SEND_DATA** VCB is passed transparently through the vendor API to Host Integration Server 2009. When the verb completes, the return code from Host Integration Server is returned to the transaction program.

3. The transaction program issues a **SYNCPT** verb to the vendor API.

4. The vendor API creates a PREPARE PS header and transmits it by issuing a **SEND_DATA** or **MC_SEND_DATA** verb. For a mapped conversation, the data_type field of the **MC_SEND_DATA** VCB must be set to AP_PS_HEADER.

5. On completion of the **SEND_DATA** or **MC_SEND_DATA** verb, the vendor API issues a RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT verb.

6. The **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb completes with the **what_rcvd** field of the VCB with a value of AP_PS_HEADER. The data buffer is filled with the received REQUEST_COMMIT PS header.

7. Another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb is issued to get send direction. Note that the vendor API can combine these two verbs into a single request by setting the **rtn_status** field of the VCB to AP_YES in order to receive status with data on the first RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT.

8. A COMMITTED PS header is then transmitted using a SEND_DATA or MC_SEND_DATA verb.

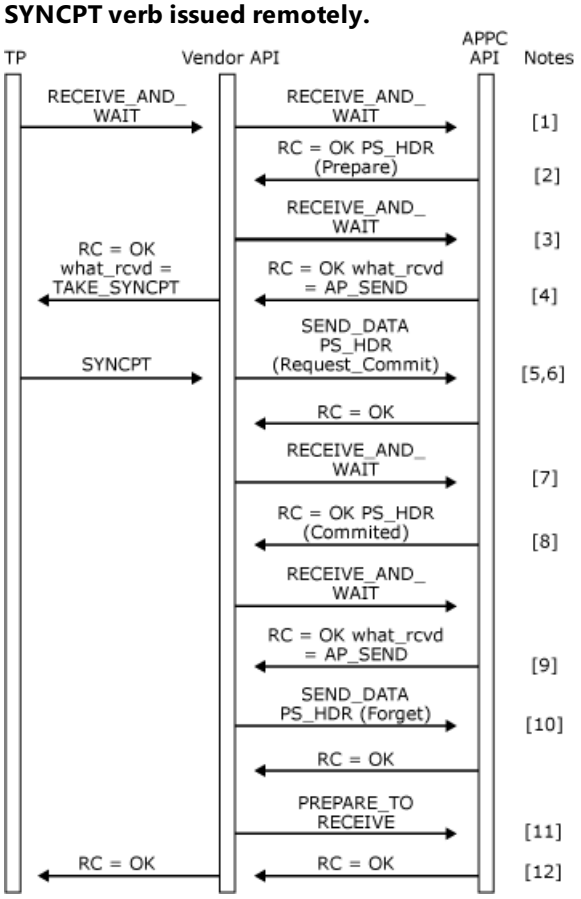
9. The Vendor API issues a **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb to receive the FORGET PS header from the remote TP.

10. Another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb is issued with the **what_rcvd** field of the VCB set to AP_SEND to get send direction (again the **rtn_status RECEIVE_AND_WAIT** field of the VCB can be set to AP_YES to combine these two verbs).

11. When send indication is received, the vendor API returns the **SYNCPT** verb to the local transaction program with an OK return code.

SYNCPT Verb Issued Remotely

SYNCPT verb issued remotely.



1. The local TP issues a RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAITverb (depending on whether a basic or mapped conversation is being used) to receive data from the remote transaction program. The vendor API passes the verb transparently to Host Integration Server.

2. The **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb completes with **what_rcvd** = AP_PS_HEADER. The data buffer contains a PREPARE PS header.

3. Another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb is issued by the vendor API to receive the send indication from the remote TP.

4. The vendor API returns the transaction program's **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb with the **what_rcvd** field of the VCB set to TAKE_SYNCPT.

5. The transaction program issues a **SYNCPT** verb.

6. The vendor API generates a REQUEST_COMMIT PS header and transmits it using a SEND_DATA or MC_SEND_DATA verb. If the conversation is mapped, the **MC_SEND_DATA** verb is issued with the **data_type** field of the VCB set to AP_PS_HEADER.

7. The vendor API then issues a RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT verb to give the remote TP direction to send.

8. The **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb completes with the **what_rcvd** field of the VCB set to AP_PS_HEADER. The data buffer contains a COMMITTED PS header.

9. Another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb is issued to get permission to send.

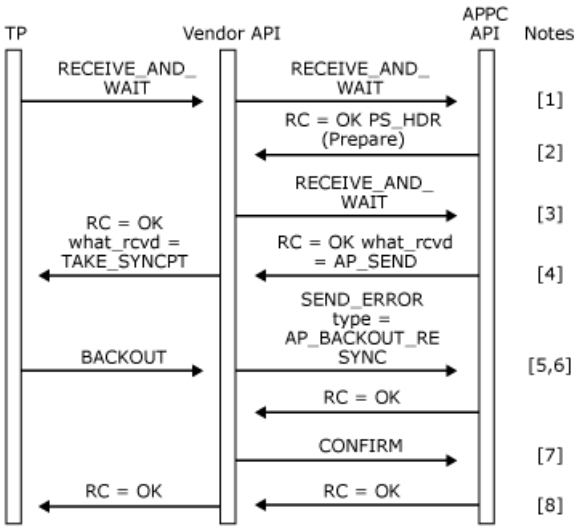
10. A FORGET PS header is prepared and sent to the remote transaction program.

11. The FORGET is flushed and direction given to the remote transaction program by issuing a PREPARE_TO_RECEIVE or MC_PREPARE_TO_RECEIVE with the **ptr_type** field of the VCB set to AP_FLUSH.

12. When the **PREPARE_TO_RECEIVE** or **MC_PREPARE_TO_RECEIVE** verb completes, the vendor API returns the **SYNCPT** verb to the local transaction program.

BACKOUT Verb Issued Locally

BACKOUT verb issued locally.



1. The local transaction program issues a RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT verb (depending on whether a basic or mapped conversation is being used) to receive data from the remote transaction program. The vendor API passes the verb transparently to Host Integration Server.

2. The **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb completes with the **what_rcvd** field of the VCB set to AP_PS_HEADER. The data buffer contains a PREPARE PS header.

3. Another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb is issued by the vendor API to receive the send indication from the remote TP.

4. The vendor API returns the transaction program's **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb with the **what_rcvd** field of the VCB set to **TAKE_SYNCPT**.

5. The transaction program issues a **BACKOUT** verb to back out the transaction.

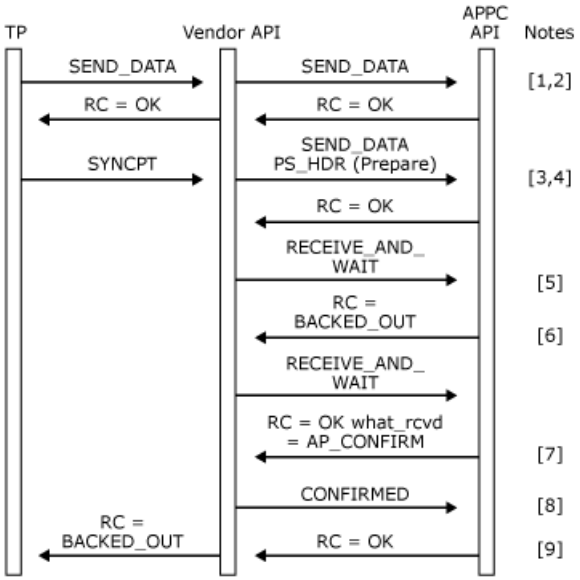
6. The vendor API generates a SEND_ERROR or MC_SEND_ERROR verb of type BACKOUT_RESYNC to send the Backout sense code 0x08240001.

7. The vendor API then issues a CONFIRM or MC_CONFIRM verb to flush the **SEND_ERROR** or **MC_SEND_ERROR** verb and request a response from the remote transaction program.

8. The CONFIRM or MC_CONFIRM verb completes when the remote transaction program issues a CONFIRMED or MC_CONFIRMED verb. The vendor API then returns the **BACKOUT** verb to the local transaction program.

BACKOUT Verb Issued Remotely

BACKOUT verb issued remotely.



1. The transaction program issues a SEND_DATA or MC_SEND_DATAverb depending on whether a basic or mapped conversation is being used.

2. The **SEND_DATA** or **MC_SEND_DATA** VCB is passed transparently through the vendor API to Host Integration Server. When the verb completes the return code from Host Integration Server is returned to the transaction program.

3. The transaction program issues a **SYNCPT** verb to the vendor API.

4. The vendor API creates a PREPARE PS header and transmits it by issuing a **SEND_DATA** or **MC_SEND_DATA** verb. For a mapped conversation, the data_type field of the **MC_SEND_DATA** VCB must be set to AP_PS_HEADER.

5. On completion of the **SEND_DATA** or **MC_SEND_DATA** verb, the vendor API issues a RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT verb.

6. The **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb returns with a return code of AP_BACKED_OUT, indicating that the remote transaction program issued a **BACKOUT** verb.

7. The vendor API issues another **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** verb to receive the Confirm indication.

8. When the verb completes with the **what_rcvd** field of the VCB set to AP_CONFIRM, the vendor API issues a CONFIRMED or MC_CONFIRMED verb to acknowledge the **BACKOUT** verb.

9. The **SYNCPT** verb is returned to the transaction program with a BACKED_OUT return code when the **CONFIRMED** or **MC_CONFIRMED** verb completes.

Windows CSV Overview

Common service verbs (CSVs) are a set of programming functions provided by Host Integration Server 2009. The CSVs provide convert, log, trace, and transfer services to applications.

The CSVs and information presented in this section represent an evolving CSV API that is composed of IBM Extended Services for OS/2 and a set of Microsoft Windows extensions that allow for registering and deregistering the application, and that provide an asynchronous entry point for TRANSFER_MS_DATA.

This section describes the verbs available to you and explains how to use them with your applications. A detailed description of each verb is provided in the reference portion of the SDK.

The CSVs are as follows:

CONVERT

Converts a character string from ASCII to EBCDIC or from EBCDIC to ASCII.

COPY_TRACE_TO_FILE

Concatenates the contents of the individual application programming interface (API)/link service trace files to form a single trace file.

DEFINE_TRACE

Enables or disables tracing for specific APIs.

GET_CP_CONVERT_TABLE

Creates and returns a 256-byte conversion table to translate character strings from a source code page to a target code page.

LOG_MESSAGE

For OS/2 only, takes a message from a message file, adds specified data to it, and records the message in the error log file. This verb optionally displays the message on the user's screen.

TRANSFER_MS_DATA

Builds a Systems Network Architecture (SNA) request unit (RU) containing Network Management Vector Transport (NMVT) data. The verb can send the NMVT data to NetView for centralized problem diagnosis and resolution. The data is optionally logged in the event log for Windows 2000.

This section contains:

- Host Integration Server Asynchronous Support
- Before Using Windows CSV
- Creating Specific NetView User Alerts
- Using CSVs in C Programs
- Sample Programs
- CSV Verb Control Block
- Bit Ordering
- WINCSV Definition
- WINCSV.H File
- Issuing a CSV

Host Integration Server Asynchronous Support

Asynchronous call completion returns the initial call immediately so the application can continue with other processes. An application that issues a call and does not regain control until the operation completes is not able to perform any other operations. This type of operation, referred to as blocking, is not suited to a server application designed to handle multiple requests from many clients.

Through **RegisterWindowsMessage** with "WinAsyncCSV" as the string, you pass a window handle by which you will be notified of call completion. You then make your call and when it completes, a message is posted to the window handle that you passed, notifying you that the call is complete.

Before Using Windows CSV

The following Microsoft® Windows® extensions are of particular importance and should be reviewed before using Windows CSV:

WinAsyncCSV

Provides an asynchronous entry point for TRANSFER_MS_DATA only. If used for any other verb, the behavior will be synchronous. Use this extension instead of the blocking version of the verb if you run your application under Microsoft® Windows® version 3.*x*.

When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinAsyncCSV" as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call. The *lParam* argument contains the original verb control block (VCB) pointer and can be dereferenced to determine the final return code.

If the function returns successfully, a "WinAsyncCSV" message is posted to the application when the operation completes or the conversation is canceled.

WinCSVCleanup

Terminates and deregisters an application from a Windows CSV implementation.

Important
An application must call this function to deregister itself from the Windows CSV implementation.

WinCSVStartup

Allows an application to specify the version of Windows CSV required and to retrieve details of the specific CSV implementation.

Important
An application must call this function to register itself with a Windows CSV implementation before issuing any further Win dows CSV calls.

Creating Specific NetView User Alerts

You can create NetView user alerts for users to send. Users identify the alerts by number; each number corresponds to a specific collection of information or requests that the user wants to send via NetView to a host operator.

Microsoft® Host Integration Server leaves blank fields for the user alert information in the structure that is returned from the sepdcrec function. To create specific user alerts, create appropriate data structures and call the TRANSFER_MS_DATA verb to send the user alert to NetView.

Using CSVs in C Programs

CSVs are available to C applications through the external function **WINCSV**.

Sample Programs

A collection of sample programs is delivered with the Microsoft® Host Integration Server Software Development Kit (SDK) in the \SDK\SAMPLES directory on the Host Integration Server CD. For more information, see APPC Samples.

CSV Verb Control Block

The only parameter passed to the **WINCSV** function is the address of a verb control block (VCB). The VCB is a structure made up of variables that identify the verb to be executed, supply information to be used by the verb, and contain information returned by the verb when execution is complete. Each verb has its own VCB structure, which is declared in the file WINCSV.H.

Bit Ordering

Bit 0 refers to the high-order bit in a byte or word. To set bit 0 on in a byte, use the bitwise **OR** operation (=) with a value of 128.

WINCSV Definition

The prototype definition of the **WINCSV** function is as follows:

```
extern void WINAPI WINCSV (LPCSV);
```

The verb control block (VCB) address parameter, a 32-bit pointer, is declared as a long integer and requires casting from a pointer to a long integer.

WINCSV.H File

Use the **#include** command to include the WINCSV.H file in any application that issues CSVs.

The WINCSV.H file, which is included with the Microsoft® Host Integration Server Software Development Kit (SDK), contains:

- The CSV function prototype.
- The structure declarations for the CSV verb control blocks (VCBs).
- The **#define** statements that substitute meaningful symbolic constants for hexadecimal values supplied to and returned by CSVs.

If a **#define** statement pertains to a hexadecimal value that is longer than one byte, a comment shows how the hexadecimal value is stored in memory.

When setting or testing CSV parameters, use the symbolic constants defined by the WINCSV.H file. When examining trace files or the contents of memory, use the hexadecimal values.

Issuing a CSV

The procedure for issuing a CSV is shown in the following sample code that uses CONVERT.

To issue a CSV

1. Create a structure variable from the verb control block (VCB) structure that applies to the verb to be issued.

```
#include <wincsv.h>
    .
    .
struct convert  conv_block;
```

   The VCB structures are declared in the WINCSV.H file; one of these structures is named **CONVERT**.

2. Clear (set to zero) the variables within the structure.

```
memset( conv_block, '\0', sizeof( conv_block ) );
```

   This procedure is not required. However, it helps in debugging and reading the contents of memory. It also eliminates the possibility that future versions of a verb are sensitive to fields that are ignored in the current version.

3. Assign values to the required VCB variables.

```
conv_block.opcode = SV_CONVERT;
conv_block.direction = SV_ASCII_TO_EBCDIC;
conv_block.char_set = SV_AE;
conv_block.len = sizeof(tpstart_name);
conv_block.source = (LPBYTE) tpstart_name;
conv_block.target = (LPBYTE) tpstart.tp_name;
```

   The values SV_CONVERT, SV_ASCII_TO_EBCDIC, and SV_AE are symbolic constants representing integers. These constants are defined in the WINCSV.H file.

   The character array TPSTART_NAME contains an ASCII string to be converted to EBCDIC and placed in the character array TPSTART.TP_NAME.

4. Invoke the verb. The only parameter is a pointer to the address of the structure containing the VCB for the verb.

```
ACSSVC((LONG) &conv_block);
```

   You can also use the following statement:

```
ACSSVC_C((LONG) &conv_block);
```

5. Use the values returned by the verb.

```
if( conv_block.primary_rc == SV_OK ) {
/* other statements */
    .
    .
    .
```

Support for APPC Automatic Logon

This section describes the support for automatic logon for Advanced Program-to-Program Communications (APPC) applications available in Host Integration Server 2009. This feature requires specific configuration by the network administrator.

The APPC application must be invoked on the local area network (LAN) side from a client of Host Integration Server. The client must be logged on to a Microsoft Windows Server 2003, Microsoft Windows XP, or Windows 2000 domain.

The client application is coded to use "program" level security, with a special hard-coded APPC user name (MS$SAME) and password (MS$SAME). When this session allocation flows from client to Host Integration Server, the server looks up the host account and password corresponding to the Windows 2000 account under which the client is logged on, and substitutes the host account information into the APPC attach message it sends to the host.

To use this feature for an APPC application, the **user_id** and **pwd** fields in the ALLOCATE or MC_ALLOCATE verbs must be hard-coded to use the string mentioned above, and **security** must be set to AP_PGM.

CPI-C Programmer's Guide

This section of the Host Integration Server 2009 Developer's Guide provides information about developing applications with the Common Programming Interface for Communications (CPI-C).

For API references and other technical information for CPI-C, see the CPI-C Programmer's Reference section of the SDK.

For sample code using CPI-C, see CPI-C Samples section of the SDK.

In This Section

Introduction to CPI-C

Common Programming Interface for Communications (CPI-C) is an application programming interface (API) that enables peer-to-peer communications among programs in a Systems Network Architecture (SNA) environment.

Through CPI-C, programs distributed across a network can work together, communicating with each other and exchanging data, to accomplish a single processing task such as querying a remote database, copying a remote file, or sending and receiving electronic mail.

The CPI-C calls and information presented in this section represent an evolving Microsoft® Windows® CPI-C that is composed of CPI-C version 1.2 and a set of Windows extensions that enable multiple applications and asynchronous call completion.

CPI-C version 1.0 was first introduced to provide a means by which two applications could speak and listen to each other; in other words, have a conversation. A conversation is the logical connection between two programs that enables the programs to communicate with each other. Programs using CPI-C converse with each other by making program calls. These calls are used to establish the full characteristics of the conversation, to exchange data, and to control the information flow between the two programs.

CPI-C version 1.1 includes four new areas of function:

- Support for resource recovery (not supported in Windows CPI-C).
- Automatic parameter conversion.
- Support for communicating with non-CPI-C programs.
- Local and remote transparency.

Built upon CPI-C version 1.1, X/Open CPI-C provided the following:

- Support for nonblocking calls.
- The ability to accept multiple conversations.
- Support for data conversion (beyond parameters).
- Support for security parameters.

CPI-C version 1.2 consolidated CPI-C version 1.1 and X/Open CPI-C and provided all the functions described previously. Windows CPI-C adds to this functionality by providing a set of extensions for asynchronous communication in addition to supporting most features in CPI-C version 1.2 with the exception of the following features:

- Full duplex operation.
- Nonblocking call behavior (as defined in the CPI-C 1.2 specification).
- Some data conversion functions.

For a complete list of unsupported functions, see CPI-C Functions Not Supported.

The use of the Windows CPI-C interface on Microsoft® Windows Server™ 2003 and Windows 2000 causes additional threads to be created within the calling process. These other threads perform interprocess communication with the SNA service over the local area network (LAN) interface that the client is configured to use (TCP/IP, IPX/SPX, or named pipes, for example).

Stopping the SNABASE service causes the application to be unloaded from memory.

This section contains:

- Windows CPI-C Asynchronous Support

- Before Using Windows CPI-C
- Using Asynchronous Call Completion
- Initial Conversation Characteristics
- Side Information for CPI-C Programs
- Configuration for CPI-C Programs
- CPI-C Considerations on Windows Server 2003, Windows XP, and Windows 2000
- Operating Systems Support for CPI-C Development
- Finding Further Information about CPI-C

Windows CPI-C Asynchronous Support

A program that issues a call and does not regain control until the call completes cannot perform any other operations. This type of operation, referred to as blocking, is not suited to a server application designed to handle multiple requests from many clients. Asynchronous call completion returns the initial call immediately, so the application can continue with other processes.

Windows Common Programming Interface for Communications (CPI-C) support is related to asynchronous communications and includes the following calls and extensions:

Set_Processing_Mode

Specify_Windows_Handle

Wait_For_Conversation

WinCPICExtractEvent

WinCPICIsBlocking

WinCPICSetBlockingHook

WinCPICSetEvent

WinCPICUnhookBlockingHook

Two methods under Microsoft Windows Server 2003 and Windows 2000 are available for asynchronous verb completion:

- Message posting using window handles.
- Waiting for Win32 events.

The traditional method uses messages posted to a window handle to notify an application of verb completion. This method was used in earlier versions of the product to support Windows 3.*x*.

Asynchronous support using message posting is appended to the Set_Processing_Mode call and enables an application to be notified of call completion on a window handle. Calling **RegisterWindowsMessage** with "WinAsyncCPIC" as the string, an application passes a window handle by which the application is notified of call completion. The application then makes the CPI-C call, and when it completes a message is posted to the window handle that was passed, notifying the application that the call is complete.

With the exception of an asynchronous Receive call that can issue certain other calls while pending, a conversation can have only one incomplete operation at any time. For more information about using an asynchronous **Receive** call, see Using Asynchronous Call Completion. In the case of an incomplete operation, the program can issue Wait_For_Conversation to test for its completion or Cancel_Conversation to end the conversation and the incomplete operation.

A second method using Win32 events for notification is supported in Host Integration Server 2009.

If an event has been registered with the conversation using **WinCPICSetEvent**, an application can call the Win32 **WaitForSingleObject** or **WaitForMultipleObjects** function to wait to be notified of the completion of the verb.

The only Windows extension functions required for Windows CPI-C are for initialization (WinCPICStartup) and termination (WinCPICCleanup) purposes. Depending on your application, other Windows extensions for handling asynchronous verb completion can be useful, but they are not required. For an example of how to use Windows CPI-C asynchronous calls and Windows extensions, see Using Asynchronous Call Completion. For a complete description of all Windows CPI-C calls and extensions, see CPI-C Calls and Extensions for the Windows Environment.

Before Using Windows CPI-C

The following Common Programming Interface for Communications (CPI-C) calls and Windows extensions are of particular importance. You should review them before using Host Integration Server 2009.

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonym. For example, **Set_Processing_Mode** is the pseudonym for a call. The actual function name is **cmspm**.

Set_Processing_Mode(**cmspm**)

Specifies for the conversation whether subsequent calls are returned when the operation they request is complete (blocking) or immediately after the operation is initiated (nonblocking). A program is notified of the completion of nonblocking calls when it issues **Wait_For_Conversation** or through a Windows message sent to a WndProc identified by *hwndNotify* in **Specify_Windows_Handle**. When the processing mode is set for a conversation, it applies to all subsequent calls on the conversation until the mode is set again.

Specify_Windows_Handle(**xchwnd**)

Sets the window handle to which a message is sent on completion of an operation in nonblocking mode.

Wait_For_Conversation(**cmwait**)

Waits for the completion of an operation that was initiated when theprocessing mode conversation characteristic was set to CM_NON_BLOCKING and CM_OPERATION_INCOMPLETE was returned in the *return_code* parameter. Use **Wait_For_Conversation** when running a background thread or a single-threaded application for Microsoft Windows Server 2003 or Windows 2000. This most likely occurs when porting code from older versions of Host Integration Server and SNA Server.

Important
An application can set the processing mode by calling **Set_Processing_Mode**. If the window handle is set to NULL, or this call is never issued, the application must call **Wait_For_Conversation** to be notified when the outstanding operation completes.

When an asynchronous operation is complete, the applications window *hwndNotify* receives the message returned by **RegisterWindowMessage** with "WinAsyncCPIC" as the input string. The *wParam* value contains the conversation return code from the operation that is completing. Its values depend on which operation was originally issued. The *lParam* argument contains the CM_PTR to the conversation identifier specified in the original function call.

WinCPICCleanup

Terminates and unregisters an application from a Windows CPI-C implementation.

Important
This function must be called by an application when finished to unregister the application from the Windows CPI-C implementation.

WinCPICExtractEvent

Provides a method for an application to determine the event handle being used for a CPI-C conversation.

WinCPICIsBlocking

Determines if a task is executing while waiting for a previous blocking call to finish. This was used when Windows version 3.*x* went into a **PeekMessageLoop** while allowing Windows to continue. Although a call issued on a blocking function appears to an application as though it blocks, the Windows CPI-C dynamic-link library (DLL) has to relinquish the processor to allow other applications to run. This means that it is possible for the application that issued the blocking call to be re-entered, depending on the messages it receives. In this instance, **WinCPICIsBlocking** can be used to determine whether the application task currently has been re-entered while waiting for an outstanding blocking call to finish.

This extension is intended to provide help to an application written to use the CM_BLOCKING characteristic of the Windows **Specify_Processing_Mode** function. **WinCPICIsBlocking** serves the same purpose as **InSendMessage** in the Windows

API.

Older applications that were originally targeted at Windows version 3.*x* and that support multiple conversations must specify CM_NONBLOCKING in **Specify_Processing_Mode** so they can support multiple outstanding operations simultaneously. Applications are still limited to one outstanding operation per conversation in all environments.

Note
Windows CPI-C prohibits more than one outstanding blocking call per thread.

WinCPICSetBlockingHook

Allows a Windows CPI-C implementation to block CPI-C function calls by means of a new function. Blocking calls apply only if you do not use asynchronous calls. If a function needs to block, the blocking call is called repeatedly until the original request completes. This allows Windows to continue to run while the original application waits for the call to return. Note that while inside the blocking call, the application can be re-entered. **WinCPICSetBlockingHook** was used by Windows version 3.*x* applications that went into a **PeekMessageLoop** to make blocking calls without blocking the rest of the system.

Note
By default, Windows Server2003 and Windows2000 do not go into a **PeekMessageLoop**. Rather, they block an event waiting for the call to complete. The only time you need to use **WinCPICSetBlockingHook** for Windows Server 2003 or Windows 2000 is when a single-threaded application for Windows Server 2003 or Windows 2000 shares common source code. In this case, you must explicitly make this call. Contrast this call with **WinCPICIsBlocking** and **WinCPICUnhookBlockingHook**.

WinCPICSetEvent

Associates a Win32 event handle with a verb completion.

WinCPICStartup

Allows an application to specify the version of Windows CPI-C required and to retrieve details of the specific CPI-C implementation.

Important
An application must call this function to register itself with a Windows CPI-C implementation before issuing any further Windows CPI-C calls.

WinCPICUnhookBlockingHook

Removes any previous blocking hook that has been installed and reinstalls the default blocking mechanism.

Using Asynchronous Call Completion

With one exception, Microsoft® Host Integration Server permits one outstanding Windows® SNA asynchronous call per connection and one blocking verb per thread. The exception to this is that when issuing an asynchronous Receive call, the following calls can be issued while the **Receive** is outstanding:

- Cancel_Conversation
- Deallocate
- Request_To_Send
- Send_Error
- Test_Request_To_Send_Received

This enables an application, in particular a 5250 emulator, to use an asynchronous **Receive** to receive data. Use of this feature is strongly recommended.

The following example illustrates how to use asynchronous call completion with Host Integration Server:

```
void ProcessVerbCompletion (WPARAM wParam LPARAM lParam)
{
    for ( i = 0; i<nPendingVerbs; i++ )
        if (memcmp  (pPending [i].ConvID, (Conversation_ID) lParam)== 0)
            ProcessCommand  (wParam, lParam);
}

LRESULT CALLBACK SampleWndProc  ( . . .)
{
    if (msg = = uAsyncCPIC ) {
        ProcessVerbCompletion (wParam, lParam);
    }
    else switch  (msg)  {
       case WM_USER:
        Initialize_Conversation (lpConvId, "GORDM", &lError  );
        if  (lError  ! = CM_OK )   {
            ErrorDisplay ( ) ;
            break ;
        }
        Set_Processing_Mode (lpConvId, CM_NON_BLOCKING, &lError  ) ;
        if  ( lError ! = CM_OK )  {
            ErrorDisplay ( ) ;
            break ;
        }
        Allocate  (lpConvId,  &lError )  ;
        switch  (lError )  {
        case CM_OK:
            break ;
        case CM_OPERATION_INCOMPLETE:
            memcopy (pPending [nPending ++].ConvId, lpConvId, sizeof (C) ;
            break ;
        default:
            ErrorDisplay ( ) ;
        }
        break ;
}
WinMain  ( . . . )
{
    if  ( ( WinCPICStartup  ( . . . )  =  =  FALSE  )   {
        return FALSE;
    }
    uAsyncCPIC = RegisterWindowMessage  ("WinAsyncCPIC"");
```

```
    Specify_Windows_Handle  (hwndSample) ;
    while  (GetMessage  ( . . . )   )   {
    . . . . .
    }
    WinCPICCleanup  ( . . . )
}
```

For more information about CPI-C calls and Windows extensions, see CPI-C Calls and Extensions for the Windows Environment. For additional information about using CPI-C, see the *IBM Systems Application Architecture Common Programming Interface Communications Reference*, part number SC26-4399-04.

Initial Conversation Characteristics

Common Programming Interface for Communications (CPI-C) maintains a set of internal values called characteristics for each conversation. Some characteristics affect the overall operation of the conversation, such as the conversation type. Others affect the behavior of specific calls, such as the receive type.

Many of these characteristics are initially derived from the side information table (see Side Information for CPI-C Programs) in memory. Initialize_Conversation specifies the symbolic destination name (*sym_dest_name*) associated with the wanted side information table entry.

The following table lists the initial values of the conversation characteristics and tells which call can change a given value.

Characteristic	**Initial value set by Initialize_Conversation**	**Initial value set by Accept_Conversation**	**Can be changed by**
Conversation state	CM_INITIALIZE_STATE	CM_RECEIVE_STATE	Depends on call
Conversation type	CM_MAPPED_ CONVERSATION	The value specified by the invoking program.	Set_Conversation_Type
Deallocate type	CM_DEALLOCATE_ SYNC_LEVEL	CM_DEALLOCATE_ SYNC_LEVEL	Set_Deallocate_Type
Error direction	CM_RECEIVE_ERROR	CM_RECEIVE_ ERROR	Set_Error_Direction
Fill	CM_FILL_LL	CM_FILL_LL	Set_Fill
Log data	Null	Null	Set_Log_Data
Log data length	0	0	Set_Log_Data
Mode name	The mode name contained in the side information. If no *sym_dest_name* is specified, this is a null string.	The mode name for the session on which the conversation startup request arrived.	Set_Mode_Name
Mode name length	Length of mode name. If no *sym_dest_name* is specified, this is zero.	Length of mode name.	Set_Mode_Name
Partner LU name	The partner logical unit (LU) name contained in the side information. If no *sym_dest_name* is specified, this is a single blank.	The partner LU name for the session on which the conversation startup request arrived.	Set_Partner_LU_Name
Partner LU name length	Length of partner LU name. If no *sym_dest_name* is specified, this is 1.	Length of partner LU name.	Set_Partner_LU_Name
Partner program name	The program name contained in the side information. If no *sym_dest_name* is specified, this is a single blank.	Not applicable.	Set_TP_Name
Partner program name length	Length of partner program name. If no *sym_dest_name* is specified, this is 1.	Not applicable.	Set_TP_Name
Password	The password contained in the side information. If no *sym_dest_name* is specified, this is a single blank.	The value specified by the invoking program.	Set_Conversation_Security_Password
Password length	Length of password. If no *sym_dest_name* is specified, this is 1.	Length of password.	Set_Conversation_Security_Password

Prepare-to-receive type	CM_PREP_TO_ RECEIVE_SYNC_ LEVEL	CM_PREP_TO_ RECEIVE_SYNC_ LEVEL	Set_Prepare_To_Receive_Type
Receive type	CM_RECEIVE_AND_ WAIT	CM_RECEIVE_AND_ WAIT	Set_Receive_Type
Return control	CM_WHEN_SESSION_ ALLOCATED	Not applicable.	Set_Return_Control
Security type	The security type contained in the side information.	The value specified by the invoking program.	Set_Conversation_Security_Type
Send type	CM_BUFFER_DATA	CM_BUFFER_DATA	Set_Send_Type
Synchronization level	CM_NONE	The value specified by the invoking program.	Set_Sync_Level
User identifier	The user identifier contained in the side information. If no *sym_dest_name* is specified, this is a single blank.	The value specified by the invoking program.	Set_Conversation_Security_User_ID
User identifier length	Length of user identifier. If no *sym_dest_name* is specified, this is 1.	Length of user identifier.	Set_Conversation_Security_User_ID

Side Information for CPI-C Programs

The information required for two Common Programming Interface for Communications (CPI-C) programs to communicate is stored as a table, called the side information table, in memory. The table is derived from the symbolic destination name (configured in Host Integration Server) and from the Set_CPIC_Side_Information, Extract_CPIC_Side_Information, and Delete_CPIC_Side_Information calls.

The side information is maintained by the system administrator.

If you are developing commercial programs or programs that will be installed on multiple computers within your organization, it is recommended that you include logic that enables a user or system administrator to specify configuration information for each copy of the program.

Each side information entry contains the following fields:

Symbolic destination name
This is the *sym_dest_name* parameter specified by Initialize_Conversation. It is the identifier for the side information entry. The name can be up to eight ASCII characters. See Set_CPIC_Side_Information for the allowed characters.

Partner LU name
This is the name by which the partner logical unit (LU) is known to the local program. It can be an alias of up to eight ASCII characters or a fully qualified network name of up to 17 characters. For the allowed characters, see Set_Partner_LU_Name.

Partner program type and name
These fields indicate whether the partner program is an application transaction program (TP) or an SNA service TP, and provide the partner program name. An application TP name can contain up to 64 ASCII characters. A service TP name can contain up to four characters. For the allowed characters, see Set_TP_Name.

Mode name
This name represents a set of characteristics to be used in an LU-to-LU session. The mode name can contain up to eight ASCII characters. For the allowed characters, see Set_Mode_Name.

Conversation security type
This field indicates whether security will be used and if so, what type.

You can use conversation security to require that the invoking program provides a user identifier and password before CPI-C allocates a conversation with the invoked program.

For an invoked program that in turn invokes another program, the security type can inform the second invoked program that security has already been verified.

For further information about conversation security, see Set_Conversation_Security_Type.

Security user identifier and password
If you intend to use conversation security, a valid combination of user identifier and password is required to access the invoked program. The user identifier and password can be up to 10 ASCII characters. For information about allowed characters, see Set_Conversation_Security_User_ID and Set_Conversation_Security_Password.

Configuration for CPI-C Programs

In addition to maintaining the side information (specified by *sym_dest_name*), the system administrator must define the following entities during configuration:

- Modes
- Local logical units (LUs)
- Partner LUs
- Invokable programs
- User identifiers and passwords

Note
For a user or group using transaction programs (TPs), 5250 emulators, or Advanced Program-to-Program Communications (APPC) applications, you can assign a default local APPC LU and a default remote APPC LU. These default LUs are accessed wh en the user or group member starts an APPC program (TP, 5250 emulator, or APPC application) and the program does not s pecify LU aliases by leaving the field NULL or filling with blanks.

CPI-C Considerations on Windows Server 2003, Windows XP, and Windows 2000

This topic summarizes information you should keep in mind when you develop programs based on Microsoft® Windows Server™ 2003, Windows® XP, and Windows 2000.

Asynchronous completion notification using message posting

When an asynchronous operation is complete, the applications window *hwndNotify* receives the message returned by **RegisterWindowMessage** with "WinAsyncCPIC" as the input string. The *wParam* value contains the *conversation_return_code* from the operation that is completing. Its values depend on which operation was originally issued. The *lParam* argument contains the CM_PTR to the *conversation_ID* specified in the original function call.

Asynchronous completion notification using Win32® events

When a verb is issued on a nonblocking conversation, it returns CM_OPERATION_INCOMPLETE if it is going to complete asynchronously. If an event has been registered with the conversation, the application can call **WaitForSingleObject** or **WaitForMultipleObjects** to be notified of the completion of the verb. WinCPICExtractEvent allows a Common Programming Interface for Communications (CPI-C) applicationto determine this event handle. After the verb has completed, the application must call Wait_For_Conversationto determine the return code for the asynchronous verb. The Cancel_Conversationfunction can be called to cancel an operation and the conversation itself.

It is the responsibility of the application to reset the event, as it is with other APIs.

If no event has been registered, the asynchronous verb completes as it does at present, which is by posting a message to the window that the application has registered with the CPI-C library.

Byte ordering

By default, Intel-byte ordering is used. For inline environments, defining NON_INTEL_BYTE_ORDER does all the required flipping for constants. Nonconstant input parameters in verb control blocks (VCBs)—for example, lengths and pointers—are always in the native format.

Events

To receive data asynchronously, an event handle is passed in the semaphore field of the VCB. This event must be in the nonsignaled state when passed to CPI-C, and the handle must have EVENT_MODIFY_STATE access to the event.

Library name

The Win32® DLL name is WINCPIC32.DLL.

Multiple threads

A transaction program (TP) can have multiple threads that issue verbs. Windows CPI-C makes provisions for multithreaded Windows-based processes. A process contains one or more threads of execution. All references to threads refer to actual threads in a multithreaded Windows environment.

Packing

For performance reasons, the VCBs are not packed. As a result, DWORDs are on DWORD boundaries, WORDs on WORDs, and BYTEs on BYTEs. VCBs should be accessed using the structures provided.

Run-time linking

For a TP to be dynamically linked to CPI-C at run time, the TP must issue:

- **LoadLibrary** to dynamically load WINCPIC.DLL or WINCPIC32.DLL, the libraries for WINCPIC.
- **GetProcAddress** to specify WINCPIC as the desired entry point to the dynamic-link library (DLL).
- **FreeLibrary** when the CPI-C library is no longer required.

Simultaneous conversations

A program can simultaneously participate in as many as 64 conversations per process.

Terminating applications

In Microsoft® Windows Server™ 2003 and Windows® 2000, CPI-C cannot tell when an application terminates. Therefore, if an application must close (for example, it receives a WM_CLOSE message as a result of an ALT+F4 from a user), the application should call WinCPICCleanup.

Yielding to other components

When processing CPI-C and Common Service Verbs (CSV), it may be necessary for the library code to yield to enable another component, such as the SnaBase, to receive messages and pass them to the application. This can be accomplished by using the Windows extensions WinCPICSetBlockingHook and WinCPICUnhookBlockingHook.

WinCPICSetBlockingHook enables a Windows CPI-C implementation to block CPI-C function calls by means of a new function. To call **WinCPICSetBlockingHook**:

```
FARPROC WINAPI WinCPICSetBlockingHook (FARPROC lpBlockFunc)
```

WinCPICUnhookBlockingHook removes any previous blocking hook that has been installed and reinstalls the default blocking mechanism. To call **WinCPICUnhookBlockingHook**:

```
BOOL WINAPI WinCPICUnhookBlockingHook (void)
```

Operating Systems Support for CPI-C Development

Host Integration Server 2009 supports the development of CPI-C applications for Microsoft Windows Server 2003, Windows XP and Windows 2000. Support for CPI-C applications is provided only for the Win32 system.

The previous Microsoft SNA Server products also supported the development of CPI-C applications for Microsoft Windows NT, Windows 98, Windows 95, Windows 3.*x*, and OS/2. Most CPI-C applications developed for Windows 3.*x* and OS/2 with SNA Server can be used with Host Integration Server.

Finding Further Information about CPI-C

For information about SNA architecture, refer to your system network documentation.

The following topics provide additional information about Host Integration Server 2009 application programming interfaces (APIs) based on SNA architecture:

- APPC Guide
- LUA Guide

For more information about SNA and about 3270 information display systems, see the following manuals:

- *IBM 3270 Information Display System: 3274 Control Unit Description and Programmers Guide*
- *IBM 3270 Information Display System: Color and Programmed Symbols*
- *IBM 3270 Information Display System: 3274 Control Unit Display Station: Operators Guide*
- *IBM Systems Network Architecture: Technical Overview*
- *IBM Systems Network Architecture: Concepts and Products*
- *IBM Advanced Communications Function Products Installation Guide*
- *IBM Installation and Resource Definition*
- *IBM 9370 LAN Token Ring Support*
- *IBM SNA Format and Protocol Reference Manual: Architectural Logic*

For background information about logical unit (LU) 6.2, Advanced Program-to-Program Communications (APPC), or CPI-C, see the following manuals:

- *IBM Systems Network Architecture: Introduction to APPC*
- *IBM Systems Network Architecture: Transaction Programmers Reference Manual for LU Type 6.2*
- *IBM SNA: Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM SNA: Formats*
- *IBM SNA: Technical Overview*
- *IBM SNA: ACF/VTAM Programming for LU Type 6.2*

CPI-C Call Summary

This section briefly describes each Common Programming Interface for Communications (CPI-C) call. The features provided by a CPI-C call can be broader than this section indicates. The calls are grouped in categories according to the function they perform. There are calls that start and stop a conversation, send and receive data, get information, and get side information.

This section contains:

Starting a Conversation

The calls in this category are used to start a conversation between two programs.

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Accept_Conversation(**cmaccp**)
Issued by the invoked program to accept the incoming conversation and set certain conversation characteristics. Upon successful execution of this call, Common Programming Interface for Communications (CPI-C) generates a conversation identifier.

Allocate(**cmallc**)
Issued by the invoking program to allocate a conversation with the partner program, using the current conversation characteristics. CPI-C can also start a session between the local logical unit (LU) and partner LU if one does not already exist. The type of conversation allocated depends on the conversation type characteristic—mapped or basic.

Initialize_Conversation(**cminit**)
Issued by the invoking program to obtain a conversation identifier and to set the initial values for the conversation's characteristics. The initial values are derived from side information associated with the symbolic destination name or are CPI-C defaults.

After issuing **Initialize_Conversation**, the invoking program can issue any of the following **Set_** calls to change the initial conversation characteristics. These calls cannot be issued after **Allocate** has been issued.

Call	**Sets**
Set_Conversation_Security_Password (**cmscsp**)	Security password
Set_Conversation_Security_Type(**cmscst**)	Conversation security type
Set_Conversation_Security_User_ID (**cmscsu**)	Security user identifier
Set_Conversation_Type (**cmsct**)	Conversation type
Set_Mode_Name (**cmsmn**)	Mode name
Set_Partner_LU_Name (**cmspln**)	Partner LU name
Set_Return_Control (**cmsrc**)	Return control
Set_Sync_Level (**cmssl**)	Synchronization level
Set_TP_Name (**cmstpn**)	Program name

Sending Data

The following calls are used to send data to the partner program:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Confirm(**cmcfm**)
Sends the contents of the local logical unit's (LU) send buffer and a confirmation request to the partner program and waits for confirmation.

Flush(**cmflus**)
Sends the contents of the local LU's send buffer to the partner LU (and partner program). If the send buffer is empty, no action takes place.

Prepare_To_Receive(**cmptr**)
Changes the state of the conversation for the local program from SEND to RECEIVE, making it possible for the local program to begin receiving data. Before changing the conversation state, this call performs the equivalent of the **Flush** or **Confirm** call.

Request_To_Send(**cmrts**)
Notifies the partner program that the local program wants to send data. The partner program may or may not act on this request.

Send_Data(**cmsend**)
Puts data in the local LU's send buffer for transmission to the partner program. The data collected in the local LU's send buffer is transmitted to the partner LU (and partner program) when one of the following occurs:

- The send buffer fills up.
- The local program issues a **Flush**, **Confirm**, or Deallocate call or other call that flushes the LU's send buffer. (Some send types, set by **Set_Send_Type**, include flush functionality.)

Set_Prepare_To_Receive_Type(**cmsptr**)
Sets the conversation's prepare-to-receive type, which specifies whether subsequent **Prepare_To_Receive** calls will include **Flush** or **Confirm** functionality. The prepare-to-receive type affects all subsequent **Prepare_To_Receive** calls. It can be changed by reissuing **Set_Prepare_To_Receive_Type**.

Set_Send_Type(**cmsst**)
Sets the conversation's send type. The send type specifies how data will be sent by **Send_Data**. The send type can specify that only data be sent or that, in addition to sending data, Common Programming Interface for Communications (CPI-C) executes the equivalent of **Flush**, **Confirm**, **Prepare_To_Receive**, or Deallocate. The send type value affects all subsequent **Send_Data** calls. It can be changed by reissuing **Set_Send_Type**.

Receiving Data

The following calls or extensions enable a program to receive data from its partner program:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Receive(**cmrcv**)
Issuing this call while the conversation is in RECEIVE state causes the local program to receive any data that is currently available from the partner program. If no data is available and the receive type is set to CM_RECEIVE _AND_WAIT, the local program waits for data to arrive. If the receive type is set to CM_RECEIVE_IMMEDIATE, the program does not wait.

Issuing this call while the conversation is in SEND or SEND_PENDING state is allowed only if the receive type is set to CM_RECEIVE_AND_WAIT. This flushes the logical unit's (LU) send buffer and changes the conversation state to RECEIVE. The local program then begins to receive data.

Set_Fill(**cmsf**)
Used in a basic conversation, this call sets the conversation's fill type, which specifies whether programs will receive data in the form of logical records or as a specified length of data. This call has an effect only in basic conversations. The fill value affects all subsequent **Receive** calls. It can be changed by reissuing **Set_Fill**.

Set_Processing_Mode(**cmspm**)
Specifies for the conversation whether subsequent calls will be returned when the operation they have requested is complete (blocking) or immediately after the operation is initiated (non-blocking). A program is notified of the completion of non-blocking calls when it issues Wait_For_Conversation or through a Microsoft® Windows® message sent to a WndProc identified by the *hwndNotify* parameter in **Specify_Windows_Handle**.

Set_Receive_Type(**cmsrt**)
Sets the conversation's receive type, which specifies whether a program issuing a **Receive** call will wait for data to arrive if data is not available. The receive type value affects all subsequent **Receive** calls. It can be changed by reissuing **Set_Receive_Type**.

Specify_Windows_Handle(**xchwnd**)
Sets the window handle to which a message is sent on completion of an operation in non-blocking mode. An application can set the processing mode by calling **Set_Processing_Mode**. If the window handle is set to NULL or this call is never issued, then the application must call **Wait_For_Conversation** to be notified when the outstanding operation completes.

WinCPICSetBlockingHook
Allows a Windows Common Programming Interface for Communications (CPI-C) implementation to block CPI-C function calls by means of a new function. This call had to be explicitly issued for Windows version 3.*x* applications to make blocking calls without blocking the rest of the system. Under Windows 2000 and later, this is not necessary.

A Windows CPI-C implementation has a default mechanism by which blocking CPI-C functions are implemented. This function enables the application to execute its own function at blocking time in place of the default function.

Confirming Receipt of Data and Reporting Errors

The following calls confirm receipt of data or report an error:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Confirmed(**cmcfmd**)
Replies to a confirmation request from the partner program. It informs the partner program that the local program has not detected an error in the received data. Because the program issuing the confirmation request waits for a confirmation, **Confirmed** synchronizes the processing of the two programs.

Send_Error(**cmserr**)
Notifies the partner program that the local program has encountered an application-level error. The local program can use **Send_Error** to inform the partner program of an error encountered in received data, to reject a confirmation request, or to truncate an incomplete logical record it is sending.

Set_Error_Direction(**cmsed**)
Specifies whether a program detected an error while receiving data or while preparing to send data. Error direction is relevant only when a program issues **Send_Error** in SEND_PENDING state—immediately after issuing Receive and receiving data as well as a *status_received* value of CM_SEND_RECEIVED.

Set_Log_Data(**cmsld**)
Used in a basic conversation, this call specifies a log message (log data) and its length to be sent to the partner logical unit (LU). This call has an effect only in basic conversations. If present, log data is sent when **Send_Error** is issued or when the conversation is abnormally deallocated. After the log data is sent, Common Programming Interface for Communications (CPI-C) resets the log data to NULL and the log data length to zero.

Getting Information

The following calls retrieve information about the characteristics of a specified conversation:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Extract_Conversation_Security_Type(**xcecst**)
Retrieves security type.

Extract_Conversation_Security_User_ID(**cmecsu**)
Retrieves security user identifier.

Extract_Conversation_State(**cmecs**)
Retrieves conversation state.

Extract_Conversation_Type(**cmect**)
Retrieves conversation type.

Extract_Mode_Name(**cmemn**)
Retrieves mode name.

Extract_Partner_LU_Name(**cmepln**)
Retrieves partner LU name.

Extract_Sync_Level(**cmesl**)
Retrieves synchronization level.

Test_Request_To_Send_Received(**cmtrts**)
Determines whether a request-to-send notification has been received from the partner program.

Ending a Conversation

The following calls end a conversation:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Deallocate(**cmdeal**)
Deallocates a conversation between two programs. Before deallocating the conversation, this call performs the equivalent of the Flush or Confirm call, depending on the current conversation synchronization level and deallocate type.

Set_Deallocate_Type(**cmsdt**)
Specifies how the conversation is to be deallocated. The deallocation instructions specified by this call take effect when **Deallocate** is issued or when the send type is set to CM_SEND_AND_DEALLOCATE and Send_Data is issued.

Administering Side Information

The following calls let CPI-C applications add, replace, retrieve, or delete side information entries from memory:

Note
The names of the calls are pseudonyms. The actual C function names appear in parentheses after the pseudonyms. For example, **Accept_Conversation** is the pseudonym for a call. The actual function name is **cmaccp**.

Delete_CPIC_Side_Information **(xcmdsi)**
Deletes side information entry.

Extract_CPIC_Side_Information **(xcmesi)**
Retrieves side information.

Set_CPIC_Side_Information **(xcmssi)**
Adds or replaces side information entry.

See Also
Reference
Side Information for CPI-C Programs

Writing CPI-C Applications

A processing task accomplished by programs using Common Programming Interface for Communications (CPI-C) is called a transaction. Consequently, programs that use CPI-C are called transaction programs (TPs). These programs communicate as peers, on an equal (rather than hierarchical) basis. The TPs use CPI-C calls to exchange status information and application data. Each TP uses CPI-C calls to supply parameters to CPI-C, which performs the preferred function and returns parameters to the TP.

TPs distributed across a local or wide area network perform distributed transaction processing.

This section describes how to write transaction programs using CPI-C and how to configure the systems on which TPs run. The topics in this section cover the following general areas:

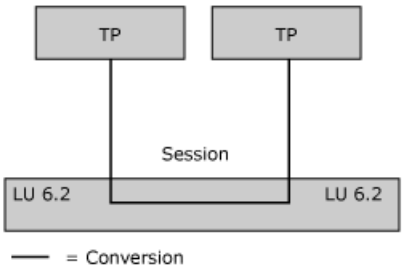
- Understanding fundamental concepts related to TPs.
- Designing and coding TPs.
- Configuring registry and environment variables for invokable TPs.
- Configuring Microsoft® Host Integration Server to work with your TPs.

This section contains:

Communication Between TPs

Various hardware and software elements in the SNA environment are required for two transaction programs (TPs) to communicate with each other. The following figure illustrates several fundamental elements.

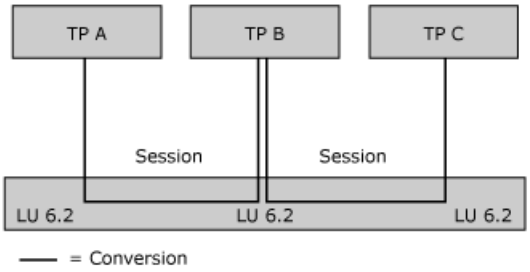
Fundamental hardware and software elements in the SNA environment



Each TP is associated with a logical unit (LU) of type 6.2. The LU enables the TP to access the network. Note that several TPs can be associated with the same LU.

A partner TP can invoke another TP, which, in turn, invokes another TP, and so on. In the following figure, TP A invokes TP B, and TP B invokes TP C.

Partner TP's invoking other partners



This section contains:

- Fundamental Terms for TPs and LUs
- Sample TPs Illustrating Fundamental Concepts
- Configuring and Controlling TPs
- Creating TPs and Their Supporting Configuration

Fundamental Terms for TPs and LUs

The following terms describe some fundamental characteristics of transaction programs (TPs) communicating through logical units (LUs):

basic conversation

A type of conversation more complex than a mapped conversation and generally used by service TPs (SNA-based programs that provide services to other programs). For a basic conversation, use Set_Conversation_Type and specify CM_BASIC_CONVERSATION for the *conversation_type*. For more information, see Basic and Mapped Conversations Compared.

conversation

The interaction between TPs carrying out a specific task. Each conversation requires an LU-LU session. A TP can be involved in several conversations simultaneously, as shown with TP B in Communication Between TPs.

invokable TP

A TP that can be invoked by another TP. Invokable TPs are usually server-type applications. That is, they work in the same general way that an IBM CICS application works. Parameters for an invokable TP are configured through registry or environment variables.

There are several types of invokable TPs:

operator-started invokable TP

A TP that is started manually in preparation for being invoked.

autostarted invokable TP

A TP that is automatically started by Common Programming Interface for Communications (CPI-C) when invoked.

queued TP

A TP that, when invoked multiple times, loads once, and then queues up subsequent requests to be dealt with one at a time. All operator-started TPs and some autostarted TPs are queued.

nonqueued TP

A TP loaded multiple times, once for every time it is invoked. Some autostarted TPs are nonqueued but no operator-started TPs are nonqueued.

For more information, see Invokable TPs.

invoking TP

A TP that can invoke (that is, initiate a conversation with) other TPs. Invoking TPs are usually client-type applications. That is, they work in the same general way that an emulator works. For more information see Invoking TPs.

local LU and local TP

An LU and TP working together, when viewed as the home base for a particular conversation. From this viewpoint, some other LU and TP are seen as the partner or remote LU and TP.

LU alias

The string that identifies an LU to a TP. The alias can be up to eight characters long.

LU-LU session

The communication between two LUs over a specific connection for a specific amount of time. An LU-LU session is needed for two TPs to interact. One session can be used serially by many pairs of TPs.

An LU 6.2 can have multiple sessions (two or more concurrent sessions with different partner LUs) and parallel sessions (two or more concurrent sessions with the same partner LU).

LUs are configured through SNA Manager on Host Integration Server 2009. This administration tool is also used to configure LU-LU pairs and modes. The LU and mode configurations control how many sessions a particular LU-LU pair supports.

mapped conversation

A type of conversation simpler than a basic conversation and generally used by application TPs (programs that accomplish tasks for end users). The default for conversation type is mapped. The conversation type can be changed with the Set_Conversation_Type call. For more information, see Basic and Mapped Conversations Compared.

partner LU and partner TP, or remote LU and remote TP

An LU and TP working together, when viewed as being at the far end of a particular conversation.

Sample TPs Illustrating Fundamental Concepts

A set of sample transaction programs (TPs) is provided on the Host Integration Server 2009 CD in the \SDK\Samples\SNA subdirectory. Included with the sample code is TPSETUP, a program that simplifies the setting of registry or environment variables needed by autostarted invokable TPs. Without an interface like that provided by TPSETUP, configuring such variables can be complicated and error-prone. Therefore, it is recommended that you use code like TPSETUP in installation programs for autostarted invokable TPs.

For information about TPSETUP and about the sample TPs, see CPI-C Samples.

Configuring and Controlling TPs

The following table shows how the characteristics of the transaction programs (TPs) and selection of the logical units (LUs) for a conversation are controlled.

Characteristic	How controlled
Type of conversation: basic or mapped.	Written into the code. For two TPs to communicate successfully, both must use the same type of conversation, basic or mapped. The default for conversation type is mapped. The type can be changed with the Set_Conversation_Type call. For more information, see Basic and Mapped Conversations Compared.
Type of TP: invoking or invokable.	Written into the code. Invoking TPs start with Initialize_Conversation and Allocate. Invokable TPs start with Accept_Conversation. For more information, see Invoking TPs and Invokable TPs.
The local LU alias to be used by an invoking TP.	Three options: • Configured with a registry or environment variable. • Configured (using SNA Manager) as the default local APPC LU for the user who starts the invoking TP. • Configured (using SNA Manager) as a member of the default outgoing local APPC LU pool. For more information, see Invoking TPs and SNA Service Configuration.
The symbolic destination name used by an invoking TP.	Written into the code, in Initialize_Conversation.
The invokable (partner) TP requested by an invoking TP.	Specified within the symbolic destination name, which can be configured using SNA Manager.
The LU alias to be used by an invokable TP (the partner LU alias from the point of view of the invoking TP).	Specified within the symbolic destination name, which can be configured through SNA Management using SNA Manager. For more information, see Invoking TPs and SNA Service Configuration and Matching Invoking and Invokable TPs.
Type of autostarted invokable TP: queued or nonqueued.	Configured with registry or environment variables. For more information, see Configuring Invokable TPs.
Local LU and remote LU aliases.	Configured using SNA Manager.
The pairing of local and remote LUs, and the mode used for each LU-LU pair.	Configured using SNA Manager.

Creating TPs and Their Supporting Configuration

The following procedure describes how to create transaction programs (TPs) and set up a supporting configuration.

To create TPs and set up a supporting configuration

1. Write, compile, and link each TP.
2. Place each TP on an appropriate computer.

   For TPs that you start many times or that are started by a user, arrange for the TP to be started easily. That is, for graphical interfaces, create a program icon for starting the TP and for non-graphical interfaces, make sure the TP is in the path.
3. On one or more computers running Host Integration Server, configure logical units (LUs), modes, LU-LU pairs, and a symbolic destination name for use by the TPs.

   For information about how to set up LU-LU pairs to support TPs, see Using Invoking and Invokable TPs.

   For information about symbolic destination names and side information, see Side Information for CPI-C Programs.
4. Set any registry or environment variables needed for the invoking and invokable TPs.

   For autostarted invokable TPs, it is recommended that you use the sample TP configuration program, TPSETUP, for this step. When you write an installation program for autostarted invokable TPs, it is recommended that you include code similar to TPSETUP.

   For information about registry or environment variables, see Configuring Invokable TPs and Invoking TPs. For information about TPSETUP, see CPI-C Samples.
5. If the invokable TP is operator-started, start it, or arrange for it to be started when the computer is restarted, and then restart the computer.

   If the invokable TP is autostarted, Host Integration Server will start it when needed.
6. Start the invoking TP.

Designing and Coding TPs

The following topics provide background information about designing and coding transaction programs (TPs).

This section contains:

CPI-C Calls in C Programs

This implementation of Common Programming Interface for Communications (CPI-C) is available to programs written in Microsoft® C version 6 or later.

The WINCPIC.H header file defines the prototypes for each CPI-C function. Other definitions include:

- Types specifically defined for use by CPI-C parameters.
- The structure of the side information entries.
- Symbolic names defined for integer parameters.

To use CPI-C calls, the C program must include WINCPIC.H and declare the variables to be used in passing parameters on CPI-C calls.

Note that you must define WIN32® before including WINCPIC.H

For example:

```
#define WIN32
#include <wincpic.h>
```

Note also that previous operating systems such as Microsoft® Windows® 98, Windows 95 and OS/2 are no longer supported.

In the case of strings, the program must also determine the preferred string length.

CPI-C and LU 6.2

Common Programming Interface for Communications (CPI-C) applications can communicate with non-CPI-C LU 6.2 applications, such as Advanced Program-to-Program Communications (APPC).

CPI-C supports all functions of logical unit (LU) 6.2 except for the following:

- Sync Point/back out processing
- PIP data
- LOCKS=LONG
- MAP_NAME
- FMH_DATA

Conversation States

The state of the conversation (as viewed by a particular transaction program (TP)) governs which Common Programming Interface for Communications (CPI-C) calls can be made by the TP at a particular time. For example, a TP cannot issue Send_Data if the conversation is not in SEND or SEND_PENDING state for that TP.

The state of a conversation depends on the TP from which it is viewed. A local TP can view a conversation as being in SEND state while the partner TP views the conversation as being in RECEIVE state. A particular TP can be in several conversations, each of which is in a different state.

The possible conversation states are summarized in this topic.

CONFIRM
The TP has received a request for confirmation of receipt of data. It must respond positively or send error information to the partner TP.

CONFIRM_DEALLOCATE
The TP has received a request for confirmation and must respond positively or send error information. If the TP responds positively, the conversation is automatically deallocated.

CONFIRM_SEND
The TP has received a request for confirmation. It must respond positively or send error information. After responding, the TP can begin to send data.

INITIALIZE
The conversation has been initialized successfully.

RECEIVE
The TP can receive application data and status information from the partner TP. When the conversation is in RECEIVE state, the TP can also send error information and request permission to send data.

RESET
The conversation has not started or has been terminated.

SEND
The TP can send data to the partner TP and request confirmation. When the conversation is in SEND state, the TP can also begin to receive data, which can cause the state to change to RECEIVE.

SEND_PENDING
The TP issued a Receive call and received data as well as a send indicator (*status_received* = CM_SEND_RECEIVED), indicating that the TP can begin to send data. This state differs from the SEND state, which occurs when the TP receives data on one **Receive** call and the send indicator on a subsequent **Receive** call.

This section contains:

- State Checks
- Changing Conversation States

State Checks

A state check occurs when a transaction program (TP) issues a Common Programming Interface for Communications (CPI-C) call and the conversation is not in the appropriate state. For example, a state check occurs if a TP issues Send_Data while the conversation is in RECEIVE state. When a state check occurs, CPI-C does not execute the call. It returns state check information through the *return_code* parameter.

Changing Conversation States

A change in the conversation state can result from:

- A call made by the local transaction program (TP).
- A call made by the partner TP.
- An error condition.

The following example shows how Common Programming Interface for Communications (CPI-C) calls can change the state of the conversation from SEND to RECEIVE and from RECEIVE to SEND.

Note
Any TP can send or receive data, regardless of whether it is the invoking TP (the TP that started the conversation) or the invokable TP (the TP that responded to a request to start a conversation).

This example shows how CPI-C calls can change the conversation state. In this table, each conversation state appears in bold and precedes the CPI-C calls that are used while in that state.

Issued by the invoking TP	**Issued by the invokable TP**
Conversation state: RESET	
Initialize_Conversation	
Conversation state: INITIALIZE	
Set_Sync_Level	
(sync_level=CM_CONFIRM)	
Allocate	
Conversation state: SEND	
Send_Data	
Prepare_to_Receive	**Conversation state: RESET**
	Accept_Conversation
	Conversation state: RECEIVE
	(status_received= CM_CONFIRM_SEND_RECEIVED)
	Conversation state: CONFIRM_SEND
	Confirm
	Conversation state: SEND
(return_code=CM_OK)	Send_Data

Conversation state: RECEIVE	Confirm
(status_received= CM_CONFIRM_RECEIVED)	
Conversation state: CONFIRM	
Request_To_Send	
Confirmed	
Conversation state: RECEIVE	(return_code=CM_OK)
	(request_to_send_received= CM_REQ_TO_SEND_RECEIVED)
	Prepare_To_Receive
Receive	
(status_received= CM_CONFIRM_SEND_RECEIVED)	
Conversation state: CONFIRM_SEND	
Confirmed	
Conversation state: SEND	(return_code=CM_OK)
	Conversation state: RECEIVE
Send_Data	
Deallocate	
	Receive
	(status_received= CM_CONFIRM_DEALLOC_RECEIVED)
	Conversation state: CONFIRM_DEALLOCATE
	Confirmed
(return_code=CM_OK)	**Conversation state: RESET**
Conversation state: RESET	

Initial States

Before the conversation is allocated, the state is RESET for both TPs.

In the example, after the conversation is allocated, the initial state is SEND for the invoking TP and RECEIVE for the invokable TP.

Changing to RECEIVE State

The Prepare_To_Receive call allows a TP to change the conversation from SEND to RECEIVE state. This call:

- Flushes the local LU's send buffer.

- Sends a CM_CONFIRM_SEND indicator to the partner TP through the *status_received* parameter of a Receive call, because the synchronization level is set to CM_CONFIRM. This indicator tells the partner TP that a Confirmed response is expected before the partner TP can begin to send data.

Changing to SEND State

The Request_To_Send call informs the partner TP (for which the conversation is in SEND state) that the local TP (for which the conversation is in RECEIVE state) wants to send data. This request is communicated to the partner TP through the *request_to_send_received* parameter of the Confirm call. (The *request_to_send_received* parameter is also returned to Send_Data and other calls.)

When the partner TP issues the Prepare_To_Receive call, the conversation state changes to RECEIVE for the partner TP, making it possible for the local TP to send data.

Important
Issuing Request_To_Send does not change the state of the conversation. Upon receiving a request to send, the partner TP is not required to change the conversation state. It can ignore the request.

Confirmation Processing

The sequence of events for confirmation processing is as follows:

1. Establish the synchronization level.
2. Send a confirmation request.
3. Receive data and confirmation request.
4. Respond to the confirmation request.
5. Deallocate the conversation.

Using confirmation processing, a transaction program (TP) sends a confirmation request with the data. The partner TP confirms receipt of the data or indicates that an error occurred. Each time the two TPs exchange a confirmation request and response, they are synchronized.

Note
Although the example in this section does not show this, any TP can send or receive data, regardless of whether the TP is the invoking TP or the invokable TP.

The following table illustrates the steps involved in confirmation processing.

Step	**Issued by the invoking TP**	**Issued by the invokable TP**
1	Initialize_Conversation	
2	Set_Sync_Level (*sync_level*=CM_CONFIRM)	
3	Allocate	
4	Send_Data	
5	Confirm	
6		Accept_Conversation
7		Receive (data_received= CM_COMPLETE_DATA_RECEIVED) (status_received= CM_CONFIRM_RECEIVED)
8		Confirmed
9	(*return_code*=CM_OK)	
10	Send_Data	
11	Deallocate	

12		Receive
13		(status_received= CM_CONFIRM_DEALLOC_RECEIVED)
14		Confirmed
15	(return_code=CM_OK)	

Establishing the Synchronization Level

The Set_Sync_Level call lets you override the default synchronization level of the conversation. The synchronization level is one of the conversation's characteristics. There are two possible synchronization levels:

- CM_CONFIRM, under which the TPs can request confirmation of receipt of data and respond to such requests.
- CM_NONE, the default, under which confirmation processing does not occur.

The Initialize_Conversation call sets the default characteristics of a conversation. There are several calls that begin with **Set**_. These calls let you override the default conversation characteristics.

Sending a Confirmation Request

Issuing the Confirm call has two effects:

- It flushes the local LU's send buffer and sends any data contained in the buffer to the partner TP.
- It sends a confirmation request that the partner TP receives through the *status_received* parameter of a Receive call.

After issuing **Confirm**, the local TP waits for confirmation from the partner TP.

Receiving a Confirmation Request

The *status_received* parameter of the Receive call indicates any future action required by the local TP.

In the example, the first Receive has a *status_received* of CM_CONFIRM_RECEIVED, indicating that a confirmation is required before the partner TP can continue.

Responding to a Confirmation Request

The partner TP issues the Confirmed call to confirm receipt of data. This frees the local TP to resume processing.

Deallocating the Conversation

Because the synchronization level of the conversation is set to CM_CONFIRM, Deallocate sends a confirmation request with the data flushed from the buffer.

For the second Receive call, *status_received* is CM_CONFIRM_DEALLOC_RECEIVED, indicating that the partner TP requires a confirmation, generated by the Confirmed call, before the conversation can be deallocated.

Conversation Security

You can use conversation security to require that the invoking transaction program (TP) provides a user identifier and password before Common Programming Interface for Communications (CPI-C) allocates a conversation with the invokable TP.

For the invoking TP, conversation security is activated and configured (with user identifier and password) through the symbolic destination name in SNA Manager or by the following calls, which override the symbolic destination name:

- Set_Conversation_Security_Type
- Set_Conversation_Security_User_ID
- Set_Conversation_Security_Password

For the invokable TP, conversation security is activated and configured through registry or environment variables on the computer where the invokable TP is located.

With communication involving more than two TPs, the verification of a user identifier and password can be passed from one TP to another. Suppose that TP A invokes TP B, which requires security information, and TP B in turn invokes TP C, which also requires security information. TP B can inform TP C that conversation security has already been verified.

For information about the registry or environment variables affecting conversation security, see Configuring Invokable TPs. For information about symbolic destination names and side information, see Side Information for CPI-C Programs.

Basic and Mapped Conversations Compared

The following table offers some guidelines for choosing between basic and mapped conversations for your transaction programs (TPs). The default for conversation type is mapped. To change to a basic conversation, use Set_Conversation_Type, and specify CM_BASIC_CONVERSATION for the *conversation_type*. For definitions of basic and mapped conversations, see Fundamental Terms for TPs and LUs.

Characteristic	**Basic conversations**	**Mapped conversations**
Common use	Generally used for service TPs.	Generally used for application TPs.
Partnering	Must be used to communicate with an existing TP that uses basic verbs.	Must be used to communicate with an existing TP that uses mapped verbs.
Sending and receiving method	Before a TP can begin a send operation, it must convert data records into logical records. The TP does this by adding a 2-byte prefix that indicates the length of the record. A TP can send several logical records at one time. When a partner TP receives logical records, it must reconstruct them into usable data records. For more information, see Logical Records Used in Basic Conversations.	A TP sends data one record at a time. Neither the sending TP nor the receiving TP needs to convert data records between different forms.
Abnormal termination	In theDeallocate call, a TP can indicate whether an error or ABEND (abnormal program termination) was caused by a TP or by a program using the TP.	A TP can indicate an error or ABEND, but cannot tell whether a problem was caused by a TP or by a program using a TP.
ABEND	A TP can indicate whether an ABEND was caused by a time-out or by a critical error.	A TP cannot indicate the cause of an ABEND.
Error logging	For an error or ABEND, a TP can send an error message, in the form of a general data stream (GDS) error log variable, to the local log and to the partner logical unit (LU).	For an error or ABEND, a TP cannot send an error message to the local log or to the partner LU.

This section contains:

- Logical Records Used in Basic Conversations
- An Example of a Mapped Conversation

Logical Records Used in Basic Conversations

Logical records are sent and received in basic conversations only.

A transaction program (TP) can send or receive multiple logical records with a single Send_Data or Receive call. A TP can also send or receive a logical record in successive portions: beginning, middle, and end.

A logical record is made up of:

- A 2-byte record-length (LL) field.
- A data field that can range in length from 0 bytes through 32765 bytes.

The LL field contains a hexadecimal value that is the length of the data field plus two bytes (for the LL field). For example, if a record contains 228 bytes of application data, the logical record length is 230. The LL field is 0x00E6, the hexadecimal equivalent of 230. If the length of the data field is 0, the value contained in the LL field is 0x0002.

Logical records are sent from or received in a data buffer established by the TP. In the data buffer, the LL field must not be in Intel byte-swapped format. For example, a length of 230 must be 0x00E6, not 0xE600.

The LL field cannot be 0x0000 or 0x0001, which allow less than the two bytes required for the LL field itself. The LL field also cannot be greater than or equal to 0x8000, which is equivalent to decimal 32768 and therefore allows for a data field greater than 32765 or an LL field greater than 2.

Setting the most significant bit of the LL field to 1 indicates that the information contained in the current logical record is continued in the next logical record.

An Example of a Mapped Conversation

The following example of a mapped conversation shows the Common Programming Interface for Communications (CPI-C) calls used to start a conversation, exchange data, and end the conversation. Call parameters are in parentheses.

Issued by the invoking TP	**Issued by the invokable TP**
Initialize_Conversation	
Allocate	
Send_Data	
Deallocate	Accept_Conversation
	Receive
	(data_received=
	CM_COMPLETE_DATA_RECEIVED)
	(*return_code*=
	CM_DEALLOCATED_NORMAL)

The following paragraphs describe the calls that are used in a mapped conversation.

Calls for Starting a Mapped Conversation

To start a conversation, the invoking transaction program (TP) issues the following calls:

- Initialize_Conversation, which requests CPI-C to set the values defining the characteristics of the conversation. The **Initialize_Conversation** call specifies a symbolic destination name that is associated with an entry in a side information table in memory. The side information specifies partner TP, partner LU, mode, security, and so on.

- Allocate, which requests that CPI-C establish a conversation between the invoking TP and the invokable TP.

The invokable TP issues the Accept_Conversation call, which informs CPI-C that it is ready to begin a conversation with the invoking TP.

Calls for Sending Data in a Mapped Conversation

The Send_Data call puts one data record (a record containing application data to be transmitted) in the send buffer of the local logical unit (LU). Data transmission to the partner TP does not happen until one of the following events occurs:

- The send buffer fills up.

- The sending TP makes a call that forces CPI-C to flush the buffer and send data to the partner TP.

In addition to the data record, the send buffer also contains the allocation request (which precedes the data record).

In the preceding example, Deallocate flushes the send buffer, sending the allocation request and data to the partner TP. Other calls that flush the buffer are Confirm and Flush.

Calls for Receiving Data in a Mapped Conversation

The **Receive** call receives the data record and status information from the partner TP. If no data or status information is currently available, the local TP, by default, waits for data to arrive.

The *data_received* parameter of Receive tells the program whether it received data and if so, whether or not the data is

complete.

Calls for Ending a Mapped Conversation

To end a conversation, one of the TPs issues Deallocate, which causes CPI-C to deallocate the conversation between the two TPs.

Using Invoking and Invokable TPs

There are two kinds of transaction programs (TPs): TPs that can invoke (that is, initiate a conversation with) other TPs, and TPs that can be invoked. A TP that can invoke another TP is called an invoking TP, and a TP that can be invoked is called an invokable TP.

The following topics describe how:

- Invoking TPs request invokable TPs.
- Invokable TPs identify themselves to the SNA service in preparation for being invoked.
- An invokable TP is matched to an invoking TP's request.

For information about how to configure logical units (LUs) to support TPs, see Configuring Host Integration Server to Support TPs.

This section contains:

Invoking TPs

An invoking transaction program (TP) can be located on any system on the SNA network. An invoking TP identifies itself by issuing Initialize_Conversation, which specifies the name of the invoking TP and the symbolic destination name to be used. A local logical unit (LU) alias can be specified for the invoking TP by using a registry or environment variable, as shown in the following table.

Operating system on computer that contains invoking TP	**Location and name of variable**
Windows 2000, Windows XP and Windows server 2003	Location in Windows 2000 registry: HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters Client <*exename*>:REG_SZ:*localLUalias* Any *exename* registry entries under the Client key represent the file names of Win32 executable files (without the file extension) for any invoking TPs. A REG_SZ value associated with each *exename* registry entry specifies the local LU alias for the invoking TP. For example, the APING.EXE Common Programming Interface for Communications (CPI-C) sample included with the Microsoft® Host Integration Server software development kit (SDK) would have the following registry entry: HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters Client APING:REG_SZ:*localLUalias*

The registry parameter for the local LU alias takes greatest precedence when associating a local LU to an invoking CPI-C application. If a registry value is not configured, two other methods are used to associate a local LU to the CPI-C application.

A local APPC LU can be associated with the user context under which the CPI-C application is running A local APPC LU can be configured by checking the **member of default local APPC LU pool** check box. Of the two possible options, a local LU associated with user context has the higher precedence.

If the local LU alias is not specified in a registry or environment variable, SNA service must be configured to supply it through one of these two types of default local LUs. Otherwise, Initialize_Conversation will fail. For more information, see Invoking TPs and SNA Service Configuration.

Next, the symbolic destination name specified in Initialize_Conversation provides the name of the invokable (or partner) TP and the partner LU alias (the LU alias to be used by the invokable TP). With this information available, the invoking TP can issue the Allocate call.

After a TP successfully issues an Allocate call, an allocation request flows. For more information about what happens after an invoking TP requests an invokable TP, see Matching Invoking and Invokable TPs.

Invoking TPs and Contention

The following information applies only to cases where logical units (LUs) are communicating in complex ways (such as chains of LUs) over multiple sessions. In such cases, two LUs may attempt to allocate a conversation on the same session at the same time. If this happens, one LU must win (the contention winner) and one must lose (the contention loser). The contention-winner LU and the contention-loser LU are determined for each session when the session is established. During that particular session, the contention-loser LU must receive permission from the contention-winner LU before allocating a conversation. In contrast, the contention-winner LU on that session allocates a conversation as needed.

Note that when two LUs are communicating over multiple sessions, one LU can be the contention winner for some of the sessions, and the other LU the contention winner for others.

An invoking transaction program (TP) operates most efficiently if the number of concurrent Allocate requests that the TP issues is matched by the number of sessions on which the local LU is the contention winner. The choice of contention winner is controlled through the modes configured at the two ends of the communication. A mode must be configured to work with the mode on the remote system for communication to begin between two LUs.

Invokable TPs

An invokable transaction program (TP) is a TP that can be invoked by another TP. Invokable TPs are written or configured through registry or environment variables to supply their names to the SNA service as a notification that they are available for incoming requests. An SNA service invokable TP can be run on any computer running Host Integration Server or client.

There are two types of invokable TPs:

Operator-started invokable TPs

An operator-started invokable TP must be started by an operator before the TP can be invoked. When the operator-started invokable TP is started, it notifies the SNA service of its availability by issuing an Accept_Conversation call. The **Accept_Conversation** call causes the name of the invokable TP to be communicated to all the SNA services in the domain, along with the alias of an associated LU if one has been configured through a registry or environment variable.

Autostarted invokable TPs

An autostarted invokable TP can be started by the SNA service when needed. The TP must be registered through registry entries or environment variables on its local system, so that it can be identified to the SnaBase component of the SNA service. The registered information defines the TP as autostarted and must specify the TP name. The registered information can also specify the local LU alias that the invokable TP will use.

The recommended method for setting registry or environment variables for autostarted invokable TPs is to use the sample TP configuration program, TPSETUP, or similar code written into your own installation program. For more information about registry or environment variables for invokable TPs, see Configuring Invokable TPs. For information about TPSETUP, see CPI-C Samples.

If no local LU alias is registered with autostarted TPs, the resulting SNA service configuration can be more flexible in responding to invoking requests. For more information about such flexible configurations, see TP Name Not Unique; Local LU Alias Unspecified.

After an autostarted invokable TP is started by SNA service, the TP issues Accept_Conversation just as an operator-started TP does. Accept_Conversation must provide the TP name that was registered for the TP.

Autostarted TPs must be configured through registry or environment variables to be either queued or nonqueued. All operator-started TPs act as queued TPs.

Queued TPs

If an autostarted TP is configured as queued, or if the TP is operator-started, incoming allocation requests are queued, and then sent only when the invokable TP issues **Accept_Conversation**. For autostarted invokable TPs, if a copy of the TP is not yet running, one is started when an incoming allocation request specifies that TP.

Note
For the Microsoft® Windows Server™2003 and Windows2000 system, only one copy of a service can be running at any gi ven time. This means that all autostarted TPs that run as services under Windows Server2003 and Windows2000 must be queued. To write an autostarted TP so it runs under Windows Server2003 and Windows2000 as a service and also runs in a nonqueued way, write a multithreaded program with an **Accept_Conversation** always outstanding.

Nonqueued TPs

If an autostarted TP is configured as nonqueued, a new copy will be started every time an Allocate is received for the TP. Nonqueued TPs should process the conversation they have been allocated, and then exit, because they will not receive any additional **Allocate** requests.

Subcategories for Invokable TPs

The following table shows subcategories for invokable transaction programs (TPs).

Queued or nonqueued	Application or service	Starting method
Queued	Running as an application or a service	Autostarted or operator-started
Nonqueued	Running as an application	Autostarted

The concept of a TP running as a service or running as an application is distinct from a service TP or an application TP. Service TP and application TP are SNA terms that describe how a TP is used: either as a supportive service program for other Common Programming Interface for Communications (CPI-C) programs, or directly by a user, as an application. For detailed information about services in Windows Server 2003 and Windows 2000, see the documentation for Windows Server 2003 and Windows 2000.

To write an autostarted TP so it runs under Windows Server 2003 and Windows 2000 as a service and also runs in a nonqueued way, write a multithreaded program with an Accept_Conversation always outstanding. For more information, see Invokable TPs.

To run an autostarted TP as an application under Windows Server 2003 and Windows 2000, make sure the TPSTART program is always started before the TP. For more information, see CPI-C Samples.

Matching Invoking and Invokable TPs

Each SNA service maintains a list of available invokable transaction program (TP) names and any logical unit (LU) aliases to be associated with the TP names. This information is obtained as follows:

- For autostarted invokable TPs, registry or environment variables identify a TP name containing a maximum of eight characters, and can specify an associated LU. This information is sent from the client to the server that sponsors the client. A client learns about the domain through a sponsor connection to a server. Clients must establish the sponsor connection before proceeding with any other tasks.

- For operator-started invokable TPs, a TP name (with a maximum of 64 characters) is specified in Specify_Local_TP_Name. The TP name is truncated to eight characters and sent from the client to the server that sponsors the client, along with the alias of an associated LU if one has been configured through a registry or environment variable.

Note
If you want a TP name to be unique, it is recommended that you limit the name to eight characters or fewer, or make th e name unique within the first eight characters. This is because the preliminary routing of allocation requests is carried out using the first eight characters. Although further matching is later carried out between the full TP names, it is ineffic ient to allow the preliminary routing to succeed when in some cases the later matching will fail.

The next step in the matching of invoking and invokable TPs is the creation of a side information table from the parameters in the symbolic destination name. Then the invoking TP issues the Allocate call and an allocation request flows to the partner LU specified in the side information table, stating the name of the invokable TP that has been requested (also listed in the side information table).

When an allocation request arrives, the SNA service compares the requested invokable TP name and LU alias to the list of available invokable TPs (which can include associated LU aliases). The comparison can be modified by registry variables, but by default is carried out as follows:

- Although the TP name requested in the symbolic destination name can be as long as 64 characters, any name received through a registry or environment variable is limited to eight characters or less. Therefore, only the first eight characters of TP names are used in comparisons.

- The comparison is carried out first on both the TP name and the LU alias. An invokable TP for which there is a match on both TP name and LU alias will be chosen ahead of a TP for which no LU alias has been configured through a registry or environment variable. A TP for which no LU alias has been configured can be matched with any request that specifies that TP name, because there cannot be a mismatch based on LU alias.

- The comparison of requested and available TP names is carried out in a specific order:

    1. The SNA service first checks for operator-started invokable TPs on the local system (the local Host Integration Server).

    2. If no match is found, the SNA service checks for autostarted invokable TPs on the local system (the local Host Integration Server).

    3. If no match is found, the SNA service checks for operator-started invokable TPs on other computers running Host Integration Server or clients.

    4. If no match is found, the SNA service checks for autostarted invokable TPs on other computers running Host Integration Server or clients.

This comparison can be modified somewhat by registry entries for the SnaServr service. The entries are called **DloadMatchTPOnly** and **DloadMatchLocalFirst**.

If a match is found, the SNA service signals the system containing the requested TP to connect to that SNA service. If no match is found, the SNA service rejects the incoming request.

For suggestions about specific ways to handle TP names and LU aliases, see Arranging TPs Within an SNA Network.

Note

Because of the way Common Programming Interface for Communications (CPI-C) works, an allocation request does not flow until local data buffers are full, or a Confirm or Flush call is made. This may mean that the allocation request does not flow until some time after the Allocate call is made. Therefore, any allocation failure caused by the rejection of the allocation request at the partner LU will be observed as the failure of a later call with one of the allocation failure return codes.

Configuring Invokable TPs

The following topics tell how to configure invokable transaction programs (TPs) for the various Microsoft® SNA service client types.

This section contains:

- Clients Running Windows XP or Windows 2000

Clients Running Windows XP or Windows 2000

On clients running Microsoft® Windows® XP or Windows 2000, invokable transaction programs (TPs) are configured through the Windows XP or Windows 2000 registry.

Note
With WindowsXP and Windows2000, the recommended method for setting registry variables for autostarted invokable TPs is to use the sample TP configuration program, TPSETUP. Compile INSTALL.C, the source code for TPSETUP, for the Windows XP and Windows2000 environment. When you write an installation program for autostarted invokable TPs, it is recommended that you add code similar to TPSETUP to the installation program. For information about TPSETUP, see CPI-C Samples.

For clients running Windows XP or Windows 2000, it is recommended that autostarted invokable TPs be written as Windows XP or Windows 2000 services. Be sure to include code like that in TPSETUP in the program that installs your TPs. Among other things, TPSETUP shows how to use the **CreateService** function when installing a TP. For important information about how services work under Windows 2000, see the documentation for Windows 2000.

The following table lists the registry entries used for the types of invokable TPs that can be run on Windows XP or Windows 2000 clients.

Type of TP	**Location in registry**	**Possible registry entries**
Autostarted invokable TP running as a service on Windows XP or Windows 2000 client.	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services *TPName*** (and subkeys)	Registry entries created by the **CreateService** call, including entries that specify the path, display name, and other characteristics of the service. —plus— Linkage OtherDependencies:REG_MULTI_SZ:SnaBase Parameters SNAServiceType:REG_DWORD:0x5 LocalLU:REG_SZ:*LUalias* Parameters:REG_SZ:*ParameterList* Timeout:REG_DWORD:*number* AcceptNames:REG_SZ:*TPNameList* ConversationSecurity:REG_SZ:{ YES | NO } AlreadyVerified:REG_SZ:{ YES | NO }*Username1*:REG_SZ:*Password1* ...*UsernameX*:REG_SZ:*PasswordX* For more information, see the notes following this table.
Autostarted invokable TP running as an application on a Windows XP or Windows 2000 client. For more information, see the notes following this table.	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters TPs *TPName*** Parameters	**SNAServiceType:**REG_DWORD:{ 0x5 | 0x6 }**PathName:**REG_EXPAND_SZ:*path***LocalLU:**REG_SZ:*LUalias***Parameters:**REG_SZ:*ParameterList***TimeOut:**REG_DWORD:*number***AcceptNames:**REG_SZ:*TPNameList***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }***Username1*:**REG_SZ:*Password1* ...***UsernameX*:**REG_SZ:*PasswordX* For more information, see the notes following this table.
Operator-started invokable TP running as a service on a Windows XP or Windows 2000 client.	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services *TPName*** (and subkeys)	Registry entries created by the **CreateService** call, including entries that specify the path, display name, and other characteristics of the service. —plus— Linkage OtherDependencies:REG_MULTI_SZ:SnaBase Parameters SNAServiceType:REG_DWORD:0x1A LocalLU:REG_SZ:*LUalias* Timeout:REG_DWORD:*number* ConversationSecurity:REG_SZ:{ YES | NO } AlreadyVerified:REG_SZ:{ YES | NO }*Username1*:REG_SZ:*Password1* ...*UsernameX*:REG_SZ:*PasswordX* For more information, see the note following this table.

Operator-started invokable TP running as an application on a Windows XP or Windows 2000 client.	**HKEY_LOCAL_MACHINE SYSTEM CurrentControlSet Services SnaBase Parameters TPs *TPName*** Parameters	**SNAServiceType:**REG_DWORD:0x1A**LocalLU:**REG_SZ:*LUalias***TimeOut:**REG_DWORD:*number***ConversationSecurity:**REG_SZ:{ YES | NO }**AlreadyVerified:**REG_SZ:{ YES | NO }***Username1*:**REG_SZ:*Password1* ...***UsernameX*:**REG_SZ:*PasswordX* For more information, see the note following this table.

Note

Before an autostarted TP can be started as an application on a Windows 2000 or later client, the TPSTART program must be started. For more information, see CPI-C Samples.

Note

AlreadyVerified and Username/Password entries are used only if ConversationSecurity is set to YES.

This section contains:

- Registry Entries for Clients Running Windows XP or Windows 2000
- Example of Registry Entries for Windows XP or Windows 2000

Registry Entries for Clients Running Windows XP or Windows 2000

The following list gives details about registry entries for clients running Windows XP or Windows 2000. For each transaction program (TP) type, the applicable variables and their locations are shown in Clients Running Windows XP or Windows 2000.

OtherDependencies:REG_MULTI_SZ:SnaBase
For a TP running as a service, ensures that the SnaBase service is started before the TP is started. This entry belongs under the **Linkage** subkey.

SNAServiceType:REG_DWORD:{ 0x5 | 0x6 | 0x1A }
Indicates the type of TP. Use a value of 0x5 for an autostarted queued TP, 0x6 for an autostarted nonqueued TP, and 0x1A for an operator-started TP.

Note that the value for an autostarted TP running as a service must be 0x5, because these TPs are always queued, as described in Invokable TPs.

PathName:REG_EXPAND_SZ: *path*
For an autostarted TP running as an application, specifies the path and file name of the TP. The data type of REG_EXPAND_SZ means that the path can contain an expandable data string For example, %SystemRoot% represents the directory containing the Windows 2000 system files. Note that for a TP running as a service, an equivalent entry is inserted by the **CreateService** call. No additional path entry is needed.

LocalLU:REG_SZ: *LUalias*
Specifies the alias of the local LU to be used when this TP is started on this computer.

Parameters:REG_SZ: *ParameterList*
Lists parameters to be used by the TP. Separate parameters with spaces.

Timeout:REG_DWORD: *number*
Specifies the time, in milliseconds, that an Accept_Conversation will wait before timing out. Specify *number* in decimal. The registry editor converts this to hexadecimal before displaying it. The default is infinity (no limit).

AcceptNames:REG_SZ: *TPNameList*
With Windows 2000, used for autostarted TPs only. Lists additional names under which the invokable TP can be invoked. Separate TP names with spaces. The default is none. If an invokable TP does not issue a Specify_Local_TP_Name for each name configured under AcceptNames in the registry, that TP will fail.

ConversationSecurity:REG_SZ:{ YES | NO }
Indicates whether this TP supports conversation security. The default is NO.

AlreadyVerified:REG_SZ:{ YES | NO }
Indicates whether this TP can be invoked with a user identifier and password that have already been verified. **AlreadyVerified** is ignored if **ConversationSecurity** is set to NO.

For a diagram of three TPs in a conversation, where the third TP can be invoked with a password that is already verified by the second TP, see Communication Between TPs. The following table shows the requirements for using password verification in a chain of TPs.

First TP (an invoking TP)	Second TP (invokable TP that confirms password, and then invokes another TP)	Third and subsequent TPs (invokable TPs that invoke other TPs)
Does not need registry or environment variables.	**ConversationSecurity** setting must be YES.	**ConversationSecurity** setting must be YES.
Does not need registry or environment variables.	**AlreadyVerified** setting can be YES or NO.	**AlreadyVerified** setting must be YES.

Symbolic destination name or **Set_Conversation_ Security_Type** in this TP specifies PROGRAM for the security type. As a result, the TP passes along the user identifier and password supplied in the symbolic destination name (or through calls (1)).	Symbolic destination name or **Set_Conversation_ Security_Type** in this TP specifies SAME for the security type. As a result, after confirming the user identifier and password, the TP passes along the user identifier and an already-verified flag.	Symbolic destination name or **Set_Conversation_ Security_Type** in this TP specifies SAME for the security type. As a result, the TP passes along the user identifier as received, along with the already-verified flag.

Note

Set_Conversation_Security_User_ID or **Set_Conversation_Security_Password** overwrites the user identifier and password specified in the symbolic destination name.

Note

If you set **AlreadyVerified** to NO, this TP cannot join in a chain of conversations where password verification is already done. (The exception to this is when **ConversationSecurity** is set to NO, in which case the TP could be the final TP in such a chain, because it performs no checking.)

Note

If you are configuring a TP that sometimes needs to confirm a password and sometimes accepts an already-verified flag, set AlreadyVerified to YES and configure the *UsernameX* variable appropriately. In this case, whenever the TP is invoked without the already-verified flag set, AlreadyVerified is ignored. Verification is attempted with the user identifier and password configured for the TP.

Note

The default for AlreadyVerified is NO. If you set AlreadyVerified to YES, make sure that ConversationSecurity is also set to YES.

Username1 :REG_SZ: *Password1*... ***UsernameX***:REG_SZ: *PasswordX*

Sets one or more user names and passwords to be compared with those sent by the invoking TP. The user name and password can each be as many as 10 characters. Both parameters are case-sensitive.

This variable is ignored if conversation security is not activated or if the password has already been verified, as described for the AlreadyVerified entry.

Example of Registry Entries for Windows XP or Windows 2000

For an autostarted invokable transaction program (TP) called **BounceTP** and running as a service, the following registry entries might be added to a client running Windows XP or Windows 2000. The entries would be added to **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services**, under the subkeys shown in bold type.

Note

In the following list, the parameters listed directly under the **BounceTP** key (such as **DisplayName** and **ErrorControl**) are service parameters created when TPSETUP or similar code is run to install the TP. These parameters should be created by TPSETUP or similar code. They should not be set manually. For more information about TPSETUP, see CPI-C Samples.

BounceTP
DisplayName:REG_SZ:BounceTP**ErrorControl:**REG_DWORD:0x1**ImagePath:**REG_EXPAND_SZ:c:\sna\system\bouncetp.exe**ObjectName:**REG_SZ:LocalSystem**Start:**REG_DWORD:0x3**Type:**REG_DWORD:0

Linkage

OtherDependencies:REG_MULTI_SZ:SnaBase

Parameters

SNAServiceType:REG_DWORD:0x5**LocalLU:**REG_SZ:JohnDoe**Parameters:**REG_SZ:Arg1 Arg2 Arg3**Timeout:**REG_DWORD:0x100**ConversationSecurity:**REG_SZ:yes**AlreadyVerified:**REG_SZ:no**JohnDoe:**REG_SZ:SecretPassword

Configuring Host Integration Server to Support TPs

The following topics describe how the Host Integration Server 2009 configuration works with invoking and invokable transaction programs (TPs.)

In This Section

Invoking TPs and SNA Service Configuration

For an SNA service to support the beginning of the invoking process, the following parameters must be configured correctly:

- If the invoking transaction program (TP) specifies the logical unit (LU) alias that it uses (in a registry or environment variable), that LU alias must match a local Advanced Program-to-Program Communications (APPC) LU alias on the supporting SNA service. If the invoking TP does not specify a local LU alias, one of two methods for designating a default LU must be carried out on the supporting SNA service:

    - Assign a default local APPC LU to the user or group that starts the invoking TP (that is, the user or group logged on at the system from which Initialize_Conversation is issued).

      —or—

    - Designate one or more LUs as members of the default outgoing local APPC LU pool. Host Integration Server or SNA service first attempts to determine the default local APPC LU of the user who started the TP, then attempts to assign an available LU from the default outgoing local APPC LU pool. If these attempts fail, SNA service rejects the request.

- In most situations, the supporting SNA service must contain an appropriate connection to another system (host or peer). Sometimes, for testing purposes, the SNA service contains two local LUs paired together (for invoking and invokable TPs that are in the same domain). In this situation, a connection to a host or peer is not necessary.

- The partner LU alias specified in the symbolic destination name must match an LU alias that is paired with the local LU alias used by the invoking TP.

The preceding parameters support the beginning of the invoking process. For the invoking process to successfully complete, additional parameters must be configured as described in Invokable TPs and the SNA Service Configuration.

Invokable TPs and the SNA Service Configuration

For an SNA service to receive allocation requests from an invoking transaction program (TP) on another system and route those requests to an invokable TP, certain parameters must be configured correctly:

- The SNA service must have a connection to the system from which the invoking TP's request is sent.

- The SNA service must have a remote logical unit (LU) capable of receiving the incoming request. This remote LU can be configured either explicitly or implicitly.

  When configured explicitly, there is an explicit match between a remote LU alias on the SNA service and the alias of the LU that conveys the invoking TP's request.

  When configured implicitly, an implicit incoming remote LU (with its implicit incoming mode) is used. This means that several items must work together. First, the LU alias specified in the incoming request (the LU alias requested for the invokable TP) must match a local LU alias on the SNA service receiving the request. Second, the local LU on the server must have an implicit incoming remote LU assigned to it. The properties of the implicit incoming remote LU will be used for that LU-LU session.

- Appropriate local LUs must be defined in the SNA service configuration. For descriptions of several ways to set up these local LUs, see Arranging TPs Within an SNA Network.

Arranging TPs Within an SNA Network

If your Host Integration Server installation contains multiple systems (clients or SNA services), you can place a given invokable transaction program (TP) on more than one system. When an invoking request is received in such an installation, there can be a choice of systems on which to run the invokable TP. You can maintain specific control over this choice. Alternatively, by following the instructions in TP Name Not Unique; Local LU Alias Unspecified, you can enable SNA service to make the choice randomly to distribute the load.

You can maintain specific control over this choice of system by setting up invokable TPs with unique names, or by setting up each invokable TP to run only with a specific, unique logical unit (LU) alias. With this arrangement, the information provided by the invoking TP (in the symbolic destination name) specifies the system on which the invokable TP should run.

You can allow the SNA service to make the system choice randomly by setting the **DloadMatchLocalFirst** registry entry to NO and using invokable TPs that leave the local LU alias unspecified. Then, when an incoming request is received, it is routed randomly, rather than preferentially to the local server. In addition, no matter what LU alias is requested for the invokable TP, there cannot be a mismatch. SNA service starts one instance of the requested TP, choosing randomly among the available systems.

The following topics describe some of the possible arrangements that can be made for running TPs.

This section contains:

- TP Name Unique for Each TP
- TP Name Not Unique; Local LU Alias Unique
- TP Name Not Unique; Local LU Alias Unspecified

TP Name Unique for Each TP

One way to specify the intended system where the invokable transaction program (TP) will run is to use a unique TP name for each invokable TP. In this arrangement, the invoking TP identifies the intended invokable TP (and system) simply by naming the TP. This makes it unnecessary for an invokable TP to specify any logical unit (LU) alias in registry or environment variables.

TP Name Not Unique; Local LU Alias Unique

Another way to specify the intended system where the invokable transaction program (TP) will run is to give the same name to multiple invokable TPs, but associate each TP with a unique local logical unit (LU) alias. To do this, configure each invokable TP (through registry or environment variables) to use a unique local LU alias. Then set up the invoking TPs so that each one is routed not only to the correct TP name but also to the correct partner LU alias for the intended invokable TP.

TP Name Not Unique; Local LU Alias Unspecified

If it does not matter on which system an invokable transaction program (TP) runs, use the same name for multiple invokable TPs and do not specify a logical unit (LU) alias in the registry or environment variables for the TPs. In this situation, there are no associated LU aliases in the list of available invokable TP names on a computer running Host Integration Server. Thus, a request received from an invoking TP cannot cause a mismatch on the LU alias, and will match according to the TP name.

In this situation, if you set the **DloadMatchLocalFirst** registry entry to NO, the SNA service randomly routes the request to one of the available TPs. This spreads the processing load among multiple systems and provides hot backup (the ability to take systems online and offline without disrupting service).

Troubleshooting for Invokable TPs

If there are difficulties with starting an invokable transaction program (TP), there may be a mismatch between the information for the invokable TP, the invoking TP, or logical units (LUs) in the SNA service configuration. That is, there may be a mismatch between the symbolic destination parameters, the registry or environment variables, or LU aliases specified in SNA Manager. For details about how to specify LU aliases in SNA Manager, see Invoking TPs and SNA Service Configuration.

Simplifying CPI-C Configuration

There are several features in Host Integration Server 2009 that can simplify configuration for Common Programming Interface for Communications (CPI-C):

- The implicit, incoming remote logical unit (LU) and the implicit, incoming mode which allow SNA service to accept requests that arrive by unrecognized remote LUs and modes.

- The default local Advanced Program-to-Program Communications (APPC) LU and the default remote APPC LU, which allow LU aliases to be associated with user or group names, simplifying the routing of incoming requests and the configuration of client systems.

- The default outgoing local APPC LU pool, which enables LUs to be allocated dynamically to any invoking TP that does not specify a local LU.

- Automatic partnering, which automatically creates LU-LU pairs and assigns modes to the pairs.

Support for CPI-C Automatic Logon

This section describes the support for automatic logon for Common Programming Interface for Communications (CPI-C) applications that is available in Host Integration Server 2009. This feature requires specific configuration by the network administrator. The CPI-C application must be invoked on the local area network (LAN) side from a client of Host Integration Server. The client must be logged into a Microsoft Windows domain, and the client application must be running on Windows Server 2003, Windows XP, or Windows 2000.).

To use this feature, the CPI-C client application is coded to use program level security, with a special hard-coded user name of MS$SAME and password of MS$SAME. When this session allocation flows from client to SNA services, Host Integration Server looks up the host account and password corresponding to the Windows 2000 account under which the client is logged on, and substitutes the host account information into the APPC attach message it sends to the host.

Use the following function calls to use CPI-C:

- Call the Set_Conversation_Security_Type function with the *conversation_security_type* parameter set to CM_SECURITY_PROGRAM.

- Call the Set_Conversation_Security_User_ID function with the *security_user_ID* parameter set to the MS$SAME string and the *security_user_ID_length* parameter set to 7.

- Call the Set_Conversation_Security_Password function with the *security_password* parameter set to the MS$SAME string and the *security_password_length* parameter set to 7.

LUA Programmer's Guide

This section of the Host Integration Server 2009 Developer's Guide provides the programmatic techniques and procedures for creating applications with the logical unit application (LUA) programming interface.

For API references and other technical information about LUA, see LUA Programmer's Reference.

For sample code using LUA, see LUA Samples.

This section contains:

- LUA Guide
- LUA Concepts
- Writing LUA Applications
- Support for LUA Single Sign-On

LUA Guide

This section provides information required to develop C-language applications that use the conventional logical unit application (LUA) programming interface to exchange data in a Systems Network Architecture (SNA) environment.

This implementation of LUA is compatible with the Request Unit Interface (RUI) and the Session Level Interface (SLI) of earlier versions of the LUA for the Microsoft Windows NT and Microsoft Windows 95 operating systems, the Windows graphical environment, and the IBM Extended Services for OS/2 version 1.0.

This section provides conceptual information and detailed reference information.

To use this section effectively, you should be familiar with:

- Microsoft Host Integration Server 2009
- Microsoft Windows Server™ 2003 and Windows 2000 Server
- SNA concepts
- General concepts for the communications software you have installed. (Refer to your product documentation for information.)
- Microsoft Visual C++ version 6.0 or later

In This Section

Operating Systems Support for LUA Development

This section contains information relating to Microsoft Windows Server 2003 and Windows 2000 Server.

Host Integration Server 2009 supports the development of logical unit application (LUA) applications for Windows Server 2003 and Windows 2000 Server. Support for LUA applications is provided only for the Win32 system.

Finding Further Information about LUA

This section does not provide a detailed explanation of products, architectures, or standards other than those directly pertaining to the Microsoft® Windows® logical unit application (LUA) programming interface. For information about specific operating environments, refer to your system documentation. For information about SNA, refer to your system network documentation.

The following topics provide additional information about Microsoft Host Integration Server application programming interfaces (APIs) based on SNA architecture:

- About the APPC Guide
- APPC Programmer's Guide
- CPI-C Programmer's Guide

For more information about SNA and about 3270 information display systems, see the following manuals:

- *IBM 3270 Information Display System: 3274 Control Unit Description and Programmers Guide*
- *IBM 3270 Information Display System: Color and Programmed Symbols*
- *IBM 3270 Information Display System: 3274 Control Unit Display Station: Operators Guide*
- *IBM Systems Network Architecture: Technical Overview*
- *IBM Systems Network Architecture: Concepts and Products*
- *IBM Advanced Communications Function Products Installation Guide*
- *IBM Installation and Resource Definition*
- *IBM 9370 LAN Token Ring Support*
- *IBM SNA Format and Protocol Reference Manual: Architectural Logic*

For background information about logical unit (LU) 6.2, Advanced Program-to-Program Communications (APPC), and/or the Common Programming Interface for Communications (CPI-C), see the following manuals:

- *IBM Systems Network Architecture: Introduction to APPC*
- *IBM Systems Network Architecture: Transaction Programmers Reference Manual for LU Type 6.2*
- *IBM SNA: Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM SNA: Formats*
- *IBM SNA: Technical Overview*
- *IBM SNA: ACF/VTAM Programming for LU Type 6.2*

LUA Concepts

The conventional logical unit application (LUA) programming interface is an application programming interface (API) that enables you to write LUA applications to communicate with host applications.

The interface is provided at the request/response unit and session levels, enabling programmable control over the Systems Network Architecture (SNA) messages sent between your communications software and the host. It can be used to communicate with any of the logical unit types 0, 1, 2, or 3 at the host. The application must send the appropriate SNA messages as required by the host.

For example, you can use LUA to write a 3270 emulation program that communicates with a host 3270 application.

This section contains:

- Windows LUA Overview
- LUs and Sessions
- Configuring for LUA
- LUA Verb Summary
- A Sample LUA Communication Sequence

Windows LUA Overview

To provide one common application programming interface (API) to port applications from various operating environments to Microsoft® Windows Server™ 2003 or Windows® 2000 Server, a Windows Systems Network Architecture (SNA) standard was created. As a direct result of this work, Windows logical unit application (LUA) was developed. The LUA verbs, routines, and information presented in this guide represent an evolving Windows LUA that is based on IBM Extended Services for OS/2 version 1.0 and includes a set of Windows extensions.

The use of the Windows LUA interface on Windows Server 2003 or Windows 2000 causes additional threads to be created within the calling process. These other threads perform interprocess communication with the SNA service over the LAN interface that the client is configured to use (for example, TCP/IP, IPX/SPX, or named pipes).

If an application using Windows LUA is running on Windows Server 2003 or Windows 2000, stopping the SNABASE service causes the application to be unloaded from memory.

This section contains:

Windows LUA Asynchronous Support

Asynchronous verb completion returns immediately from issuing an initial verb (before results have been received) so the application can continue with other processes. A program that issues a verb and does not regain control until the operation completes cannot perform any other operations. This synchronous type of operation, called blocking, is not suited to a server application designed to handle multiple requests from many clients.

By design, logical unit application (LUA) is asynchronous and uses semaphores for notification messages. Semaphores work well for Windows Server 2003 or Windows 2000. Windows LUA provides the following functions for issuing the Request Unit Interface (RUI) and Session Level Interface (SLI) verbs:

- RUI
- SLI
- WinRUI
- WinSLI

WinRUI and **WinSLI** provide asynchronous message notification for all Windows-based RUI and SLI verbs, while **RUI** and **SLI** provide support for event notification. Windows version 3.*x* applications use **WinRUI** and **WinSLI** for asynchronous message notification.

Asynchronous support allows you to be notified of verb completion based on a window handle. You can register a window handle using the RegisterWindowsMessage function with "WinRUI" or "WinSLI" as the string. You then issue a verb using the WinRUI or WINSLI function and passing a window handle. When the LUA verb conversation completes, a message is posted to the window handle that you passed, notifying you that the verb is complete.

The only other Windows extension functions required for Windows LUA are for initialization (WinRUIStartup or WinSLIStartup) and termination (WinRUICleanup or WinSLICleanup) purposes. Depending on your application, other Windows extensions may be useful, but they are not required. A complete description of all Windows LUA verbs, routines, and extensions is provided in LUA RUI Verbs, LUA SLI Verbs, and LUA Extensions for the Windows Environment.

Before Using Windows LUA

The following Windows extensions are of particular importance and should be reviewed before using the logical unit application (LUA) application programming interface (API) and this version of Host Integration Server 2009:

- RUI

  Provides event notification for all Request Unit Interface (RUI) verbs. The application must provide a handle to an event in the **lua_post_handle** member of the verb control block (VCB). The event must be in the not-signaled state. When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

- SLI

  Provides event notification for all Session Level Interface (SLI) verbs. The application must provide a handle to an event in the **lua_post_handle** member of the VCB. The event must be in the not-signaled state. When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

- WinRUI

  Provides asynchronous notification for all Windows-based RUI verbs. When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinRUI" as the input string. The *lParam* argument of the message contains the address of the VCB being posted as complete. The *wParam* argument of the message is undefined.

  An application must call WinRUIStartup for initialization before calling WinRUI.

- WinRUICleanup

  An application must call this function when finished using RUI verbs to deregister itself from the Windows LUA implementation. This function terminates and deregisters an application from a Windows LUA implementation.

- WinRUIStartup

  An application must call this function to register itself with a Windows LUA implementation before issuing any further Windows LUA calls using RUI verbs. This function allows an application to specify the version of Windows LUA required and to retrieve details of the specific LUA implementation.

- WinSLI

  Provides asynchronous notification for all Windows-based SLI verbs. When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinSLI" as the input string. The *lParam* argument of the message contains the address of the VCB being posted as complete. The *wParam* argument of the message is undefined.

  An application must call WinSLIStartup for initialization before calling WinSLI.

- WinSLICleanup

  An application must call this function when finished using SLI verbs to deregister itself from the Windows LUA implementation. This function terminates and deregisters an application from a Windows LUA implementation.

- WinSLIStartup

  An application must call this function to register itself with a Windows LUA implementation before issuing any further Windows LUA calls using SLI verbs. This function allows an application to specify the version of Windows LUA required

and to retrieve details of the specific LUA implementation.

Using LUA and Asynchronous Verb Completion

Host Integration Server permits one outstanding Windows SNA asynchronous call per connection and one blocking verb per thread. When the asynchronous verb completes, logical unit application (LUA) does the following for a Windows Server 2003 or Windows 2000 system.

Two types of notification are possible:

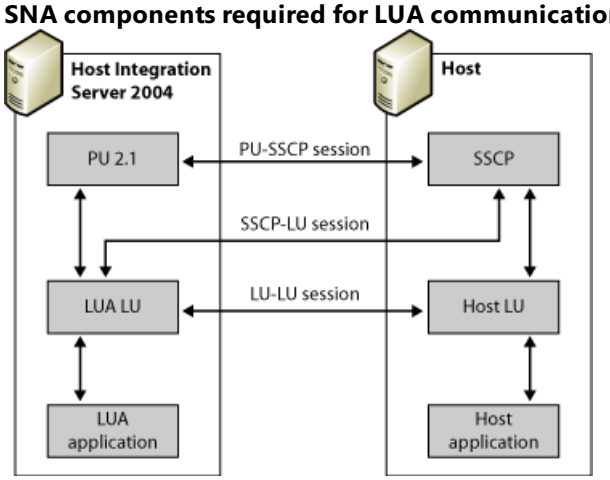
- The preferred type is the event method, in which the LUA application issues **WaitForSingleObject/WaitForMultipleObject**.

- The application can also post the"WinRUI"/"WinSLI" notification message to the window handle of the **WinRUI/WinSLI** message.

LUs and Sessions

The following figure shows the SNA components required for logical unit application (LUA) communications.

SNA components required for LUA communications



An LUA application uses a local LU, which uses Host Integration Server 2009 to communicate with the host system. There are three progressive sessions when Host Integration Server connects to the host node:

- The PU-SSCP session, between the Host Integration Server physical unit (PU) and the host's system services control point (SSCP). This is used mainly for diagnostic information. LUA communications require only the capabilities of PU 2.0. Host Integration Server provides these capabilities, plus the additional capabilities included in PU 2.1.
- The SSCP-LU session, between the LUA LU at the computer and the SSCP. This is used for controlling the LU.
- The LU-LU session, between the LUA LU at the computer and the host LU. This is used for data transfer between the computer and the host application.

LUA allows applications to send and receive data on the SSCP-LU session and on the LU-LU session. An LUA application can send data on this session using the common service verb TRANSFER_MS_DATA. LUA does not provide access to the PU-SSCP session.

The SSCP and LU sessions each provide two priorities of messages, normal and expedited. Expedited messages take precedence over other messages waiting to be transmitted on the same session. There are four different flows on which a message can be sent or received:

- SSCP session (expedited flow)
- LU session (expedited flow)
- SSCP session (normal flow)
- LU session (normal flow)

The LU session normal flow carries most of the data. The other flows are used only for control purposes.

Note
The implementation of LUA in Host Integration Server 2009 does not allow applications to send data on the SSCP expedited flow and does not return data to an application on this flow.

Configuring for LUA

The Host Integration Server 2009 configuration file, which is set up and maintained by the system administrator, contains information that is required for logical unit application (LUA) applications to communicate. An LUA LU is configured by the link service to use a connection to the host, and is given an LU number that matches that of an LU on the host.

The configuration can include LUA LU pools. A pool is a group of LUs with similar characteristics, and it enables an application to use any free LU from the pool. This feature can be used to allocate LUs on a first-come, first-served basis when there are more applications than LUs available, or to provide a choice of LUs on different connections.

The following communications components are configured for use with an LUA application.

Component	**Description**
Link service	A link service for communicating with the host. This component is normally configured by the Setup program during Host Integration Server 2009 installation.
Connection	A connection to the host that uses the link service.
Local node	A local node that owns the connection. This component is configured automatically.
LUA LU	An LUA LU on the local node, configured to use the connection, with an LU number that matches an LU on the host.
LUA LU pool (optional)	If necessary, you can configure more than one LUA LU for the application, and group the LUs into a pool. This means that an application can specify the pool rather than a specific LU when issuing the RUI_INIT verb to start a session. Host Integration Server 2009 uses this name in one of the following ways: • If the name supplied is the name of an LU that is not in a pool, a session is assigned using that LU if it is available (that is, if it is not already in use by an LUA application). • If the name supplied is the name of an LU pool, or the name of any LU within the pool, a session is assigned using the first available LU in the pool (if one is available). Note that this may not be the LU whose name was specified by the RUI_INIT verb.

LUA Verb Summary

Logical unit application (LUA) application programs can establish and use SNA sessions with either the Request Unit Interface (RUI) application programming interface (API) or the Session Level Interface (SLI) API. If an LUA application establishes an SNA session using **RUI_INIT**, it cannot issue any SLI verbs for that session. Likewise, if an LUA application establishes an SNA session using **SLI_OPEN**, it cannot issue any RUI verbs for that session.

Following is a brief summary of each LUA verb or user-supplied routine. Each verb supplies parameters to LUA, which performs the desired function and returns parameters to the application:

RUI_BID
: Allows the application to determine when information from the host is available to be read.

RUI_INIT
: Sets up the SSCP-LU session for an LUA application.

RUI_PURGE
: Cancels an outstanding **RUI_READ**.

RUI_READ
: Receives data or status information sent from the host to the LUA application's LU, on either the SSCP session or the LU session.

RUI_TERM
: Ends the SSCP session for an LUA application. It also terminates the LU session if it is active.

RUI_WRITE
: Sends data to the host on either the SSCP session or the LU session.

SLI_BID
: Notifies the SLI application that a message is waiting to be read using **SLI_RECEIVE**. It also provides the current status of the session to the LUA application.

SLI_BIND_ROUTINE
: An optional, user-supplied exit routine that notifies the LUA application that a BIND request has come from the host. It allows the routine to examine the request and formulate a response.

SLI_CLOSE
: Ends a session opened with **SLI_OPEN**.

SLI_OPEN
: Transfers control of the specified LU to the LUA application. It establishes a session between the SSCP and the specified LU, as well as an LU-LU session.

SLI_PURGE
: Cancels **SLI_RECEIVE** verbs issued with a wait condition.

SLI_RECEIVE
: Receives responses, SNA commands, and data into the buffer of an LUA application. It also provides the current status of the session to the LUA application.

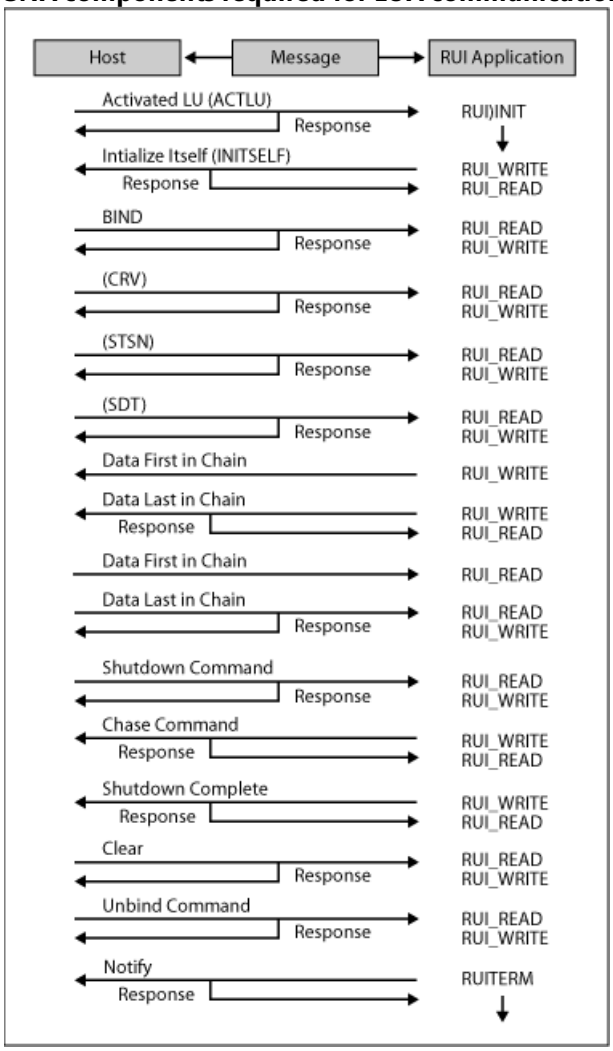
SLI_SEND
: Sends responses, SNA commands, and data from an LUA application to a host LU.

SLI_STSN_ROUTINE
: An optional, user-supplied exit routine that notifies the LUA application that a set and test sequence number (STSN) command has come from the host. It allows the routine to examine the request and formulate a response.

A Sample LUA Communication Sequence

This section illustrates how Request Unit Interface (RUI) and Session Level Interface (SLI) verbs are used for a logical unit application (LUA) communication sequence. The two figures illustrate the LUA verbs used to start a session, to exchange data, and to end the session, as well as the SNA messages sent and received. The arrows indicate the direction in which SNA messages flow.

Communication Sequence Using RUI Verbs
SNA components required for LUA communications



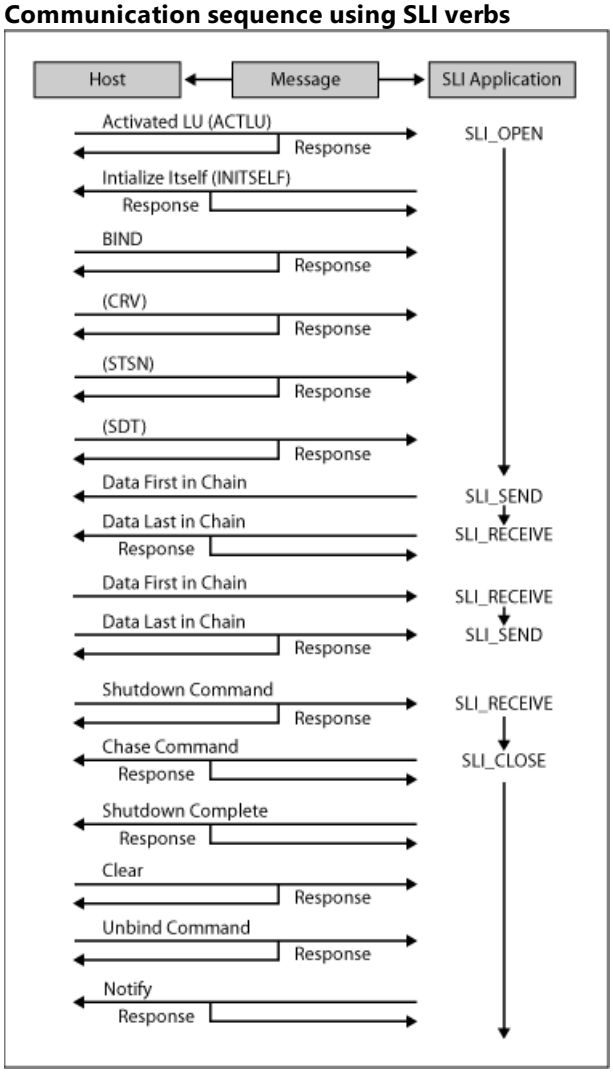
In this example, the application performs the following tasks:

- Issues an RUI_INIT verb to establish the system services control point (SSCP) session. (**RUI_INIT** does not complete until the LUA application has received an ACTLU message from the host and sent a positive response. However, these messages are handled by Host Integration Server 2009 and are not exposed to the LUA application.)
- Sends an INITSELF message to the SSCP to request a BIND and reads the response.
- Reads a BIND message from the host and writes the response. This establishes the LU session.
- Reads an SDT message from the host, which indicates that initialization is complete and data transfer can begin.
- Sends a chain of data consisting of three request/response units (RUs) and reads the response. The last RU indicates that a definite response is required.
- Reads a chain of data consisting of three RUs and writes the response.
- Reads an UNBIND message from the host and writes the response. This terminates the LU session.

- Issues RUI_TERM to terminate the SSCP session. (Host Integration Server sends a NOTIFY message to the host and waits for a positive response. However, these messages are handled by Host Integration Server and are not exposed to the LUA application.)

Communication Sequence Using SLI Verbs

Communication sequence using SLI verbs



In the example shown here, the application performs the following tasks:

- Issues an SLI_OPEN verb to establish the SSCP session.
- Sends an INITSELF message to the SSCP to request a BIND and reads the response.
- Reads a BIND message from the host and writes the response. This establishes the LU session.
- Reads an SDT message from the host, which indicates that initialization is complete and data transfer can begin.

> **Note**
> INITSELF, BIND, and SDT messages are handled by Host Integration Server if the application is using SLI. The **SLI_OPEN** does not return until Host Integration Server has sent an SDT and response.

- Issues SLI_SEND and SLI_RECEIVE to transfer data, SNA commands, or SNA responses between the host and the application.
- Issues SLI_CLOSE to terminate the SSCP session. (Host Integration Server sends a NOTIFY message to the host and waits for a positive response. However, these messages are handled by Host Integration Server and are not exposed to the LUA application.)

Writing LUA Applications

The information contained in this section will help you write logical unit application (LUA) application programs for use with Microsoft® Host Integration Server.

This section contains:

- Using LUA Verbs
- LUA VCB Format
- LUA Synchronous and Asynchronous Verb Completion
- Compiling and Linking an LUA Application
- Resetting LUA LUs
- Multiple Processes and Multiple Sessions Using LUA
- Programming Techniques for LUA Pools
- Writing Portable LUA Applications
- LUA System Considerations on Microsoft Windows Server 2003 or Windows 2000
- SNA Considerations Using LUA

Using LUA Verbs

This implementation of logical unit application (LUA) is available to applications written in Microsoft® C++® version 6.0 or later. Applications access all LUA functions on Microsoft Windows™ Server 2003 or Windows® 2000 Server by issuing verbs using the external C functions RUI,SLI,WinRUI, and WinSLI.

Symbolic constants are defined in the WINLUA.H header file for many parameter values. Refer to the WINLUA.H file (contained in the Microsoft Host Integration Server SDK) for a list of LUA constants.

You should use the symbolic constant and not the hexadecimal value when setting values for supplied parameters, or when testing values of returned parameters.

Parameters marked as reserved should always be set to zero.

This section contains:

- RUI and SLI Definitions
- Using an LUA Verb

RUI and SLI Definitions

The definitions of the RUI and SLI functions are as follows:

```
void WINAPI RUI(struct LUA_VERB_RECORD FAR * verb);
void WINAPI SLI(struct LUA_VERB_RECORD FAR * verb);
int WINAPI WinRUI(HWND handle, struct LUA_VERB_RECORD FAR * verb);
int WINAPI WinSLI(HWND handle, struct LUA_VERB_RECORD FAR * verb);
```

The WINLUA.H header file supplied with your Host Integration Server SDK includes prototypes of these functions.

The only parameter passed to the **RUI** or **SLI** function is the address of a verb control block (VCB). The VCB is a structure made up of variables that:

- Identify the logical unit application (LUA) verb to be executed.
- Supply information used by the verb.
- Contain information returned by the verb when execution is complete.

The parameters passed to the WinRUIor WinSLI function are a window handle and the address of a VCB. The window handle is used for message notification when the issued verb has completed.

The VCB structure is declared in the WINLUA.H header file. For general VCB information, see LUA VCB Format. For verb-specific VCB information, see the reference documentation for each verb.

Issuing an LUA Verb

The following procedure is required to issue a logical unit application (LUA) verb. In this example, the verb issued is RUI_INIT.

To issue an LUA verb

1. Create a variable for the verb control block (VCB) structure. For example:

```
#include <winlua.h>
        .
        .
struct LUA_VERB_RECORD rui_init;
```

2. The LUA_VERB_RECORD structure is declared in the WINLUA.H header file.

3. Clear (set to zero) the variables within the VCB:

```
memset( &rui_init, 0, sizeof( rui_init) );
```

   LUA requires that all reserved parameters, and all parameters not required by the verb being issued, must be set to zero. The simplest way to do this is to set the entire VCB to zeros before setting the parameters required for this particular verb.

4. Assign values to the VCB parameters that supply information to LUA:

```
rui_init.common.lua_verb = LUA_VERB_RUI;
rui_init.common.lua_verb_length = sizeof(struct LUA_COMMON);
rui_init.common.lua_opcode = LUA_OPCODE_RUI_INIT;
memcpy (rui_init.common.lua_luname, "THISLU  ", 8);
```

   The values LUA_VERB_RUI and LUA_OPCODE_RUI_INIT are symbolic constants. These constants are defined in the WINLUA.H header file in the Host Integration Server SDK. To ensure portability between different systems, use symbolic constants and not integer values.

5. Invoke LUA. The only parameter is a pointer to the address of the structure containing the VCB for the desired verb.

```
RUI( &rui_init );
```

6. Check the asynchronous flag (**rui_init.common.lua_flag2.async**) to determine whether the verb completed asynchronously. If events are being used and the verb did complete asynchronously, wait for the event to complete.

```
if (rui_init.common.lua_flag2.async)
{
/* verb will complete asynchronously so continue
with other processing */
/* then wait */
WaitForSingleObject (...)
}
```

   Do not check the return code. It may have changed from LUA_IN_PROGRESS to LUA_OK by the time you check it.

7. Check the variables returned by LUA.

```
if( rui_init.common.lua_prim_rc == LUA_OK )
{
/* Init OK */
```

```
        .
        .
}
else
{
/* Do error routine */
        .
        .
}
```

LUA VCB Format

The logical unit application (LUA) verb control block (VCB) is called LUA_VERB_RECORD. It is a structure with two parts:

- A structure, LUA_COMMON, which is used for all the LUA verbs.
- A union, LUA_SPECIFIC, which is used only by RUI_BID,SLI_BID,SLI_OPEN, and SLI_SEND.

This section contains:

- LUA_VERB_RECORD
- LUA_COMMON
- LUA_SPECIFIC

LUA_VERB_RECORD

The logical unit application (LUA) verb control block (VCB) structure is as follows:

```
typedef struct LUA_VERB_RECORD {
    struct LUA_COMMON common;
    union LUA_SPECIFIC specific;
} LUA_VERB_RECORD;
```

Remarks

To access parameters in the common part of the VCB, you need to include the structure member name **common**. For example, when using a verb record structure named **Lua_Verb**, you access its **lua_prim_rc** member as **Lua_Verb.common.lua_prim_rc**.

To access parameters in the specific part of the VCB, you need to include the union member name **specific.** For example, when issuing RUI_BID using a verb record structure named **Lua_Verb**, you access its **lua_peek_data** member as **Lua_Verb.specific.lua_peek_data**.

For a complete listing of the structures and related values in the LUA VCB, see LUA Verb Control Blocks.

LUA_COMMON

The following structure lists the common data structure parameters used by all the logical unit application (LUA) verbs.

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
} LUA_COMMON;
```

Remarks

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs or LUA_VERB_SLI for Session Level Interface (SLI) verbs. For both of these macros the value is 0x5200.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, for example, LUA_OPCODE_RUI_BID for the RUI_BID verb.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_OPEN and RUI_INIT require this parameter. Other Windows LUA verbs only require this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Specifies the offset from the start of the VCB to the extension list of user-supplied dynamic-link libraries (DLLs). Not used by RUI in Host Integration Server and should be set to zero.

lua_cobol_offset

Offset of the COBOL extension. Not used by LUA in Host Integration Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Specifies the length of received buffer for RUI_READand SLI_RECEIVE. For other RUI and SLI verbs, it is not used and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of data returned in **lua_peek_data** for the RUI_BID verb.

lua_data_ptr

Pointer to the application-supplied buffer that contains the data to be sent for SLI_SEND and RUI_WRITE or that will receive data for **SLI_RECEIVE** and **RUI_READ**. For other RUI and SLI verbs, this parameter is not used and should be set to zero.

lua_post_handle

Supplied parameter. Used under Windows Server 2003 or Windows 2000 if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message sent or received. Various sub parameters are set for write functions and returned for read and bid functions.

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits.

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for the write function and returned by the read and bid functions.

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits.

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. This parameter is used by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, SLI_RECEIVE, and SLI_SEND. For other LUA verbs, this parameter is not used and should be set to zero.

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

SSCP expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. This is a returned parameter for RUI_INIT and SLI_OPEN and a supplied parameter for SLI_SEND. For other LUA verbs, this variable is not used and should be set to zero.

Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

The SLI receives and responds to the BIND, CRV, and STSN requests through the LUA interface extension routines.

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. This parameter is returned by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, SLI_RECEIVE, and SLI_SEND. For other LUA verbs this parameter is not used and should be set to zero.

lua_flag2.bid_enable

Indicates that **RUI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Supplied parameter. Reserved field used by SLI_OPEN and RUI_INIT. For all other LUA verbs, this parameter is reserved and should be set to zero.

lua_encr_decr_option

Field for cryptography options. On **RUI_INIT**, only the following are supported:

- **lua_encr_decr_option** = 0
- **lua_encr_decr_option** = 128

For all other LUA verbs, this parameter is reserved and should be set to zero.

LUA_SPECIFIC

The following union shows the specific data structure that is included for functions that use the **LUA_SPECIFIC** part of a verb control block. The only logical unit application (LUA) verbs that use this union are RUI_BID,SLI_BID,SLI_OPEN, and SLI_SEND.

```
union LUA_SPECIFIC {
    struct SLI_OPEN open;
    unsigned char lua_sequence_number[2];
    unsigned char lua_peek_data[12];
} LUA_SPECIFIC;
```

Remarks

Members

open

The union member of **LUA_SPECIFIC** used by the **SLI_OPEN** verb.

lua_sequence_number

The union member of **LUA_SPECIFIC** used by the **SLI_SEND** verb. Returned parameter. Sequence number of the RU to the host.

lua_peek_data

The union member of **LUA_SPECIFIC** used by the **RUI_BID** and **SLI_BID** verbs. Returned parameter. Contains up to 12 bytes of the data waiting to be read.

LUA_SPECIFIC.SLI_OPEN

The following structure shows the **SLI_OPEN** fields of the **LUA SPECIFIC** union member for the SLI_OPEN verb.

```
struct SLI_OPEN {
    unsigned char lua_init_type;
    unsigned char lua_resv65;
    unsigned short lua_wait;
    struct LUA_EXT_ENTRY lua_open_extension[3];
    unsigned char lua_ending_delim;
} SLI_OPEN;
```

Remarks

Members

lua_init_type

Type of session initiation.

lua_resv65

Reserved field.

lua_wait

Secondary retry wait time.

lua_open_extension

Supplied parameter. Specifies any user-supplied dynamic-link libraries (DLLs) used to process specific LUA messages.

lua_ending_delim

Extension list delimiter.

LUA_EXT_ENTRY

The following structure shows the **LUA_EXT_ENTRY** fields of the LUA SPECIFIC.SLI_OPEN union member for the SLI_OPEN verb.

```
struct LUA_EXT_ENTRY {
     unsigned char lua_routine_type;
     unsigned char lua_module_name[9];
     unsigned char lua_procedure_name[33];
};
```

Remarks

Members

lua_routine_type

Extension routine type.

lua_module_name

Extension DLL module name.

lua_procedure_name

Procedure name to call in the extension DLL module.

LUA Synchronous and Asynchronous Verb Completion

Logical unit application (LUA) verbs can complete execution either synchronously or asynchronously.

Synchronous Verb Completion

When LUA is able to complete all the processing for a verb as soon as it is issued, the verb has completed synchronously. When this happens, the primary return code is set to a value other than LUA_IN_PROGRESS, and the **lua_flag2.async** bit is set to zero.

The value of the **lua_flag2.async** bit should be tested, not the primary return code being not equal to LUA_IN_PROGRESS. (For information about these returned parameters, see individual verb descriptions.)

Asynchronous Verb Completion

Some LUA verbs (for example, RUI_PURGE) complete quickly after local processing. However, most verbs take some time to complete because they require messages to be sent to and received from the local node or the host application.

When LUA must wait for information from the remote LU or from the local node before it can complete a verb, the verb completes asynchronously.

When this happens, the **lua_flag2.async** bit is set to 1. The primary return code is also normally set to LUA_IN_PROGRESS, but this value cannot be relied on. The value of the **lua_flag2.async** bit should be tested. The application can now perform other processing, or wait for notification from LUA that the verb has completed. LUA issues this notification by setting the primary return code to its final value and leaving **lua_flag2.async** set to 1.

When the verb completes, LUA does the following depending on your environment:

- For Windows Server 2003 or Windows 2000, two types of notification are possible. The LUA application either:

  Issues **WaitForSingleObject** or **WaitForMultipleObject**.

  —or—

  Posts the "WinRUI/WinSLI" notification message to the window handle of the **WinRUI**/**WinSLI** message.

  The event method using **WaitForSingleObject** or **WaitForMultipleObject** is the preferred way to receive asynchronous notification on Windows Server 2003 or Windows 2000.

- In the Windows environment, it notifies the completion of an asynchronous request by posting the "WinRUI/WinSLI" notification message to the window handle of the **WinRUI**/**WinSLI** message. A window handle has been added as the first parameter passed to the WinRUI and WinSLI entry points.

Compiling and Linking an LUA Application

Use the following procedure to compile and link a logical unit application (LUA) application.

To compile and link an LUA application for use with Host Integration Server

1. Update the path statement to include the directory containing the LUA application.
2. Set any required environmental variables.
3. Compile the application, including the WINLUA.H header file provided in the Host Integration Server SDK, to produce the .obj files.
4. Link the application with the WINLUA.LIB library to produce an .exe file.

Resetting LUA LUs

Microsoft® Host Integration Server provides a facility for resetting logical unit application (LUA) LUs or forcing off LUA applications, which is useful if an application has become deadlocked or is looping.

The NetView command **deactivate-oldlu** can also be used to reset an LUA LU. These facilities interact with the LUA application as described in the following paragraphs.

When an LUA LU is reset through Host Integration Server or by using the **deactivate-oldlu** command, Host Integration Server sends an UNBIND message to the application (as though the host had issued it).

The UNBIND message sent to the application is 0x32 0x0E, indicating a recoverable LU failure, and is returned to the application on a subsequent RUI_READ. The LU session is terminated, but the system services control point (SSCP) session remains active. (The LU is returned to the same state as if RUI_INIT has just completed.)

Multiple Processes and Multiple Sessions Using LUA

Two processes cannot use the same logical unit application (LUA) session. Only the process that issues RUI_INIT can use the session that is started by the verb. Before another process can use LUA, it must issue **RUI_INIT** to obtain a new session. However, different threads of the same process can issue verbs for the same LUA session.

A single process can simultaneously use more than one LUA session by issuing multiple **RUI_INIT** verbs. Win32® processes support for up to 15,000 sessions for applications based on the Microsoft® Windows Server™ 2003 or Windows® 2000 Server. Each session must use a different LU. Two or more sessions can use the same pool, but the **lua_luname** member (which is either the name of the pool or the name of an LU within the pool) must be different for each **RUI_INIT**.

Two or more instances of the same LUA application can run as different processes, but they must use different LUs. This can be done by using LU pools. The two processes can specify the same pool, but are allocated different LUs from that pool.

Programming Techniques for LUA Pools

When working with a logical unit application (LUA) LU pool, specify the pool name at the beginning of the conversation and then use the **lua_sid** member (not the pool name) with subsequent calls. This is necessary because the **lua_sid** member is a unique identifier, but the pool name is not, because a pool is designed to supply LUs for multiple conversations.

When using RUI_INIT or SLI_OPEN with an LUA pool, specify the pool name with the **lua_luname** member. For subsequent calls in the same conversation, use the **lua_sid** member returned from **RUI_INIT** or **SLI_OPEN** to specify the conversation.

Note
On completion of RUI_INIT or SLI_OPEN, the lua_luname member contains the actual name of the LU used. This allows you t o create code for a display for the user, showing the actual LU name used in a particular conversation.

Writing Portable LUA Applications

Use the following guidelines for writing logical unit application (LUA) applications that are portable to other environments:

- Use the symbolic constant names for parameter values and return codes, and not the numeric values shown in the WINLUA.H file. (For more information, see the WINLUA.H file in the Microsoft® Host Integration Server SDK.)

- When accessing SNAsense codes in a data buffer, use the symbolic constants rather than the numeric values. This ensures that the byte storage order is correct for your particular system. You should use **memcpy** to set the values, and **memcmp** to test them. For example:

```
memcpy (this_verb.common.lua_data_ptr, LUA_INCORRECT_REQ_CODE, 4);
if (memcmp (this_verb.common.lua_data_ptr,
LUA_INCORRECT_REQ_CODE, 4) == 0)
{
.....
}
```

- Ensure that any parameters shown as reserved are set to zero.

- Set the **lua_verb_length** parameter as described in the verb description.

LUA System Considerations on Microsoft Windows Server 2003 or Windows 2000

This section provides specific information about developing logical unit application (LUA) applications for the following operating systems.

This section contains:

- LUA Considerations on Microsoft Windows Server 2003 or Windows 2000

LUA Considerations on Windows Server 2003 or Windows 2000

This topic summarizes information for developing Win32 logical unit application (LUA) applications for Microsoft Windows Server 2003 or Windows 2000 Server.

Byte ordering

The values of constants defined in the WINLUA.H file are dependent on the byte ordering of the hardware used. Macros are used to set the constants to the correct value.

Currently, the include files in Windows Server 2003 or Windows 2000 to indicate the hardware. These same macros are used by Host Integration Server 2009, along with the Win32 macro, to indicate the byte ordering needs. The macros must be defined in the application or on the command line when building the application.

For example, the primary return code of LUA_PARAMETER_CHECK is defined to have a value of 0x0001. Depending on the environment, the constant LUA_PARAMETER_CHECK may or may not be 0x0001. Some formats define the value as it appears in memory. Others define it as a 2-byte variable. Because it cannot be assumed that an application will always use provided constants rather than hardwired values, a macro can be defined to swap the bytes. The following example shows how the macro can be used:

```
#define LUA_PARAMETER_CHECK LUA_FLIPI (0X0001)
```

Events

To receive data asynchronously, an event handle is passed in the semaphore field of the verb control block (VCB). This event must be in the non-signaled state when passed to LUA, and the handle must have EVENT_MODIFY_STATE access to the event.

Library names

To support the coexistence of Win16 and Win32 API libraries on the same computer, the Win32 dynamic-link library (DLL) names were changed from the names used by Win16 API libraries. Win32 stub DLL libraries using the old names are supplied with Host Integration Server 2009 so that older applications will still run, although these platforms are no longer supported.

Old DLL names	New DLL names
WINRUI.DLL	WINRUI32.DLL
WINSLI.DLL	WINSLI32.DLL

The new DLL names should be used for all new applications intended to run on Host Integration Server.

Load-time linking

To be dynamically linked to LUA at load time, you must do one of the following at link time:

- Insert the following **IMPORTS** statements in the definition (.def) file used to link the application:

  (For RUI)

  ```
  IMPORTS WINRUI.RUI
  IMPORTS WINRUI WinRUI
  IMPORTS WINRUI.WinRUIStartup
  IMPORTS WINRUI.WinRUICleanup
  ```

  (For SLI)

  ```
  IMPORTS WINSLI.SLI
  IMPORTS WINSLI.WinSLI
  IMPORTS WINSLI.WinSLIStartup
  IMPORTS WINSLI.WinSLICleanup
  ```

- Link the application to WINRUI.LIB (for RUI) or WINSLI.LIB (for SLI), which contain the entry-point linkage information.

Multiple threads

An LUA application can have multiple threads that issue verbs. LUA for the Win32 system makes provisions for multithreading processes on Windows Server 2003 or Windows 2000. A process contains one or more threads of execution. All references to threads refer to actual threads in a multithreaded Windows Server 2003 or Windows 2000.

Packing

For performance considerations, the VCBs are not packed. VCB structure member elements after the first element are aligned on either the size of the member type or DWORD boundaries, whichever is smaller. As a result, DWORDs are aligned on DWORD boundaries, WORDs are aligned on WORD boundaries, and BYTEs are aligned on BYTE boundaries. This means, for example, that there is a 2-byte gap between the primary and secondary return codes. Therefore, the elements in a VCB should only be accessed using the structures provided.

This option for structure and union member alignment is the default behavior for Microsoft C/C++ compilers. For compatibility with the supplied LUA libraries, make sure to use an equivalent structure and union member packing option when using other C/C++ compilers or when explicitly specifying a structure alignment option when using Microsoft compilers.

Registering and deregistering applications

All LUA applications for the Windows Server 2003 or Windows 2000 system must call the Windows SNA extension WinRUIStartup or WinSLIStartup at the beginning of the session to register the application and WinRUICleanup or WinSLICleanup at the end of the session to deregister the application.

Restrictions on 3270-style LUs

A Windows Server 2003 or Windows 2000 process cannot access 3270-style LUs from both the function management interface (FMI) and LUA APIs at the same time. However, the process can use the LUA APIs to access LUA LUs while using FMI APIs to access 3270-style LUs.

Run-time linking

For an application to be dynamically linked to LUA at run time, it must issue the following calls:

- **LoadLibrary** to load the specified library module for Windows LUA. That is, WINRUI32.DLL (for RUI), and WINSLI32.DLL (for SLI).

- **GetProcAddress** to retrieve the address of the LUA function entry points exported by the DLL. For RUI, the function entry points are RUI, WinRUI, WinRUIStartup, and WinRUICleanup. For SLI, the function entry points are SLI, WinSLI, WinSLIStartup, and WinSLICleanup.

SNA Considerations Using LUA

This section explains SNA information you need to consider when writing logical unit application (LUA) applications.

BIND checking

During initialization of the LU session, the host sends to the LUA application a BIND message that contains information such as request/response unit (RU) sizes for use by the LU session. Microsoft® Host Integration Server returns this message to the LUA application on RUI_READ. The LUA application must verify that the parameters specified on the BIND are suitable. The application has the following options:

- It can accept the BIND as it is, by issuing RUI_WRITE containing an OK response to the BIND. No additional BIND data can be sent on the response.
- It can try to negotiate one or more BIND parameters. (This is only permitted if the BIND is negotiable.) To do this, the application issues **RUI_WRITE** containing an OK response, but including the modified BIND as data.
- It can reject the BIND by issuing **RUI_WRITE** containing a negative response, using an appropriate SNA sense code as data.

  Validating the BIND parameters and ensuring that all messages sent are consistent with them is the responsibility of the LUA application. However, the following two restrictions apply:

- Host Integration Server rejects any **RUI_WRITE** that specifies an RU length greater than the size specified on the BIND.
- Host Integration Server requires the BIND to specify that the secondary LU is the contention winner and that error recovery is the responsibility of the contention loser.

Note
For SLI, an application must specify that it will use SLI_BIND_ROUTINE on the SLI_OPEN if it will do any BIND checking.

Courtesy acknowledgments

Host Integration Server keeps a record of requests received from the host to correlate any response sent by the application with the appropriate request. When the application sends a response, Host Integration Server correlates the response with the data from the original request, and can then free the storage associated with it.

If the host specifies exception response only (a negative response can be sent but a positive response should not be sent), Host Integration Server must still keep a record of the request in case the application subsequently sends a negative response. If the application does not send a response, the storage associated with this request cannot be freed.

Because of this, Host Integration Server enables the LUA application to issue a positive response to an exception-response-only request from the host. (This is known as a courtesy acknowledgment.) The response is not sent to the host, but is used by LUA to clear the storage associated with the request.

Note
The application does not need to send a courtesy acknowledgment for each exception-response-only request. For efficiency, t he application can respond less frequently. The node treats a courtesy acknowledgment as an implicit acknowledgment for al l prior pending requests.

Distinguishing SNA sense codes from other secondary return codes

A secondary return code that is not a sense code always contains a value of zero in its first two bytes.

An SNA sense code always contains a nonzero value in its first two bytes. The first byte gives the sense code category and the second identifies a particular sense code within that category. (The third and fourth bytes can contain additional information or can be zero.)

Information on SNA sense codes

If you need information about a returned sense code, see

- Status-Error Message
- Error and Sense Codes

Negative responses and SNA sense codes

SNA sense codes can be returned to an LUA application in the following cases:

- When the host sends a negative response to a request from the LUA application, it includes an SNA sense code indicating the reason for the negative response. This is reported to the application on a subsequent RUI_READ or SLI_RECEIVE with the following information.

Sense code	Description
Primary return code	LUA_OK.
Request/response indicator, response type indicator, and sense data included indicator	All set to 1, indicating a negative response that includes sense data.
Data returned	The SNA sense code.

- When Host Integration Server receives invalid data from the host, it generally sends a negative response to the host and does not pass the invalid data to the LUA application. This is reported to the application on a subsequent RUI_READ,SLI_RECEIVE, RUI_BID,orSLI_BID with the following information:

Sense code	Description
Primary return code	LUA_NEGATIVE_RESPONSE.
Secondary return code	The SNA sense code sent to the host.

- In some cases, Host Integration Server detects that data supplied by the host is invalid, but cannot determine the correct sense code to send. In this case, it passes the invalid data in an exception request (EXR) to the LUA application on RUI_READ or SLI_RECEIVE with the following information.

Sense code	Description
Request/response indicator	Set to 0, indicating a request.
Sense data included indicator	Set to 1, indicating that sense data is included. (This indicator is normally used only for a response.)
Message data	A suggested SNA sense code.

The application must then send a negative response to the message. It can use the sense code suggested by Host Integration Server, or it can alter the sense code.

- Host Integration Server can send a sense code to the application to indicate that data supplied by the application was invalid. This is reported to the application on RUI_WRITE or SLI_SEND with the following information.

Sense code	Description
Primary return code	LUA_UNSUCCESSFUL.
Secondary return code	SNA sense code.

The sense codes that can be returned as secondary return codes on LUA verbs are listed in the WINLUA.H header file. For this file, see the Host Integration Server or SNA SDK.

Pacing

Pacing is handled by the LUA interface. An LUA application does not need to control pacing and should never set the pacing indicator flag.

If pacing is being used on data sent from the LUA application to the host (determined by the BIND), RUI_WRITE or SLI_SEND may take some time to complete. This is because LUA has to wait for a pacing response from the host before it can send more data.

If an LUA application transfers large quantities of data in one direction, either to the host or from the host (for example, a file transfer application), the host configuration should specify that pacing is used in that direction. This ensures that the node receiving the data is not flooded with data and does not run out of data storage.

Purging data to end of chain

When the host sends a chain of request units to an LUA application, the application can wait until the last RU in the chain is received before sending a response, or it can send a negative response to an RU that is not the last in the chain. If a negative response is sent mid-chain, LUA purges all subsequent RUs from this chain and does not send them to the application.

When LUA receives the last RU in the chain, it indicates this to the application by setting the primary return code of RUI_READ or RUI_BID to LUA_NEGATIVE_RESPONSE with a zero secondary return code.

The host can terminate the chain by sending a message such as CANCEL while in mid-chain. In this case, the CANCEL message is returned to the application on RUI_READ. The LUA_NEGATIVE_RESPONSE return code is not used.

Segmentation

Segmentation of RUs is handled by the LUA interface. LUA always passes complete RUs to the application, and the application should pass complete RUs to LUA.

Support for LUA Single Sign-On

This section describes the logical unit application (LUA) support for Single Sign-On using 3270 display sessions that is available in Host Integration Server 2009.

Over 3270 LUs, a Single Sign-On feature for LUA applications is supported to automate the overall logon process. When configured for this feature, Host Integration Server 2009 automatically replaces special keywords in the data stream with the actual host user name and password at appropriate points in the session.

Note
Single Sign-On is not supported over LUA logical units (LUs).

To open 3270 LUs from an LUA application using Request Unit Interface (RUI), the **lua_resv56[1]** field must be set to a nonzero value when this verb control block (VCB) is passed to RUI_INIT. To open 3270 LUs from an LUA application using a Session Level Interface (SLI), the **lua_resv56[2]** field must be set to a nonzero value when this VCB is passed to SLI_OPEN. For details, see the reference sections in RUI_INIT and SLI_OPEN.

This section contains:

- Prerequisites for LUA Single Sign-On
- Registry Settings Used for LUA Single Sign-On
- LUA User Name and Password Replacement

Prerequisites for LUA Single Sign-On

In preparation for using Single Sign-On over 3270 logical units (LUs), the system administrator must define a host security domain containing host connections. This host security domain must be initially created or modified to enable the Single Sign-On feature. The system administrator must enable a users Microsoft® Windows Server™ 2003 or Windows® 2000 Server account in the host security domain and either the administrator or the user must establish a mapped host account for the Windows Server 2003 or Windows 2000 domain user name.

The user must be logged on to a Windows Server 2003 or Windows 2000 domain with a user name and password. Note that this Single Sign-On feature is only supported over 3270 LUs.

Registry Settings Used for LUA Single Sign-On

The logical unit application (LUA) Single Sign-On feature depends on Host Integration Server 2009 scanning 3270 logical units (LUs) used in the logon process for special keywords that are defined in the registry on the computer running Host Integration Server 2009. The values for these special keywords can be defined by the system administrator on the computer running Host Integration Server 2009.

The registry settings used by the LUA Single Sign-On process are located under the **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services** registry node. Installed under the **SNASERVR\PARAMETERS** subkey are the following entries:

3270SSOPadByte

This entry should be set to an ASCIIZ string to use as the character for padding replacement text in the user name or password if these strings are shorter than the length of the special tag strings. The default value for this pad character is the ASCII space character.

3270SSOPostReplaceCount

This entry should be set to a DWORD that represents the number of message chains of request/response units (RUs) to scan after replacement of text for user name or password. The default value for this number is 10.

3270SSOPrefix

This entry should be set to an ASCIIZ string to use as the special prefix tag string in combination with the user name and password tags. The default value of this string is MS$.

3270SSOPwdTag

This entry should be set to an ASCIIZ string to use as the special tag string in combination with the **3270SSOPrefix** tag in defining the special host password string that will be replaced. The default value of this string is SAMEP, so the default host password string that is scanned for and replaced is MS$SAMEP. Note that length of the password string that is scanned for (MS$SAMEP, for example) determines the maximum length of the password string that can sent to the host using Single Sign-On. This limit occurs because the password substitution cannot change the length of the data message. Note that the value of this string must be different from the value of the **3270SSOUserTag** entry for Single Sign-On to function properly.

3270SSOReplaceCount

A DWORD value that affects the time-out value for password substitution. User IDs and passwords will be substituted in each chain on the LU-SSCP and PLU-SLU sessions until the timer expires. By default the timer will be set to 30 seconds, but this behavior can be reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node.

If the **3270SSOReplaceCount** registry entry is defined and the **3270SSOReplaceTimer** registry entry is not defined, the node counts this number of RUs (on PLU-SLU session only) before time-out occurs. If both the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries are defined, the value for **3270SSOReplaceCount** will be used to determine when a time-out will occur. By default, this key is not defined, and the node defaults to a time-out of 30 seconds.

3270SSOReplaceTimer

A DWORD value that affects the time-out value for password substitution. User IDs and passwords will be substituted in each chain on the LU-SSCP and PLU-SLU sessions until the timer expires. By default the timer will be set to 30 seconds, but this behavior can be reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node.

If the **3270SSOReplaceTimer** registry entry is defined and **3270SSOReplaceCount** is not defined, the node uses this value in seconds before time-out occurs. If both the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries are defined, the value for **3270SSOReplaceCount** will be used to determine when a time-out will occur. By default, this key is not defined and the node defaults to a time-out of 30 seconds.

3270SSOUserTag

This entry should be set to an ASCIIZ string to use as the special tag string in combination with the **3270SSOPrefix** tag in defining the special user name string that will be replaced. The default value of this string is SAMEU, so the default user name string that is scanned for and replaced is MS$SAMEU. Note that length of the user name string that is scanned for (MS$SAMEU, for example) determines the maximum length of the user name string that can sent to the host using Single Sign-On. This limit occurs because the user name substitution cannot change the length of the data message. Note that the value of this string must be different from the value of the **3270SSOPwdTag** entry for Single Sign-On to function properly.

LUA User Name and Password Replacement

The SNA node on the host monitors the inbound session for a replacement sequence consisting of the **3270SSOPrefix** string immediately followed by one of the strings **3270SSOUserTag** or **3270SSOPwdTag**. Thus, the default user name string that would be scanned for and replaced is MS$SAMEU. When this string is found in the inbound session data, the node looks up the corresponding information (host user name in the current host security domain) and overwrites MS$SAMEU with this information. The same process occurs for the password string that would be scanned for and replaced, which defaults to MS$SAMEP.

Note that this operation cannot change the length of the data message. If the actual user name or password that is retrieved from the current host security domain is shorter than the replacement sequence, it is padded out with the first character of the **3270SSOPadByte** string used as a padding character. If the actual host user name or password string is longer than the string that is scanned for, these strings are truncated to the length of the scanned string so that the data message length is not affected.

Note that because the user name and password can be sent in any order, the registry string values for the **3270SSOUserTag** and **3270SSOPwdTag** entries must be different for Single Sign-On to function properly.

The SNA node monitors the SSCP-LU session for these special tag strings at all times and replaces all occurrences of these strings with corresponding looked-up data. On the LU-LU session, the node starts monitoring at start of session (BIND). The node stops monitoring when it has received **3270SSOPostReplaceCount** chains of request/response units (RUs) without seeing a substitution tag. The node will not restart monitoring until it receives an UNBIND–BIND sequence for that session.

Note that the node considers the sequence:

```
BIND, data, UNBIND(BIND FORTHCOMING), BIND
```

as a continuation of the same LU-LU session and does not restart monitoring on receipt of the second BIND. This sequence is often used by host session managers handing off a session to an application system, and is considered a single terminal session.

User IDs and passwords will be substituted in each chain on the LU-SSCP and PLU-SLU sessions until the node has received **3270SSOPostReplaceCount** chains of RUs without seeing a substitution tag or a timer expires. By default the timer is set to 30 seconds, but this behavior can be reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node. After the timer expires, the node will stop scanning messages for the 3270 replacements strings for the user ID and password. If the replacement strings arrive after the timer expires, the replacement strings will be sent to the host unmodified causing the sign-on to fail. The application will not receive any notification that the timer has expired. The only indication of a problem will likely be that the sign-on to the host session has failed.

Note

All strings are specified in the registry in ASCII, but the node translates them to EBCDIC through AE character mapping before scanning for a match.

3270 Emulation Programmer's Guide

This section provides information for independent software vendors who are developing their own 3270 Emulator client software to work with Microsoft Host Integration Server 2009 or Microsoft SNA Server.

To use this section of the guide effectively, you should be familiar with:

- Host Integration Server 2009
- Microsoft Windows Server™ 2003 or Windows® 2000
- System Network Architecture (SNA) concepts

Operating systems support for 3270 development

Host Integration Server 2009 supports the development of 3270 client applications for Windows Server 2003, Windows XP, and Windows 2000.

Network operating systems support for 3270 development

Host Integration Server 2009 supports the following network operating systems:

- Native TCP/IP
- Novell NetWare

For API references and other technical information about the 3270 Emulator, see the3270 Emulator Programmer's Reference section of the SDK.

In This Section

- Host Integration Server Concepts for 3270 Client Access
- DL-BASE/DMOD Interface
- Function Management Interface
- FMI Status, Error, and Sense Codes
- Configuration Information
- Compiling and Linking 3270 Client Applications
- Support for 3270 Single Sign-On

Host Integration Server Concepts for 3270 Client Access

This section describes some key concepts used in Host Integration Server 2009 when providing 3270 client access. The purpose of this section is to provide information that enables independent software vendors to integrate their 3270 emulators with Host Integration Server. Only the relevant parts of the Host Integration Server architecture are described.

In This Section

Structure of 3270 Client Access Components

The components of Host Integration Server 2009 that apply to 3270 client access are:

- Local nodes
- Link services
- 3270 emulation program

This section introduces the structure of these components and explains specialized terms.

In This Section

Role of the Base

The Base is a part of each Host Integration Server gateway component, such as the local 2.1 node or a link service that provides the operating environment for the core functions of that component. It passes messages between components and provides functions common to all components, such as diagnostic tracing.

The Base type used with Host Integration Server 3270 emulation programs is DL-BASE. The Host Integration Server DL-BASE supports a single Host Integration Server component or a single user application, and has entry points for initialization, sending messages, receiving messages, and termination.

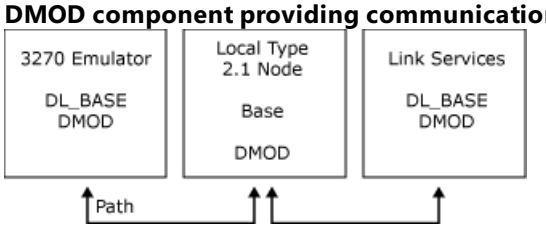
Localities and DMODs

A Base and the components within it are called a locality. The Host Integration Server system consists of one or more communicating localities, all running Host Integration Server executable programs within the local area network (LAN) Manager domain. For each Host Integration Server system, there is a single configuration file.

In a system such as Host Integration Server 2009, where the number of localities and their types are not configured in advance, the relationships between the localities are set up dynamically. Localities that can enter and leave a system in this way are called dynamic localities. Dynamic localities can enter or leave the system at any time.

Dynamic localities communicate using the Dynamic Access Module (DMOD) component, which provides the communications facilities needed to pass messages between the Bases. The following figure shows a system consisting of three dynamic localities.

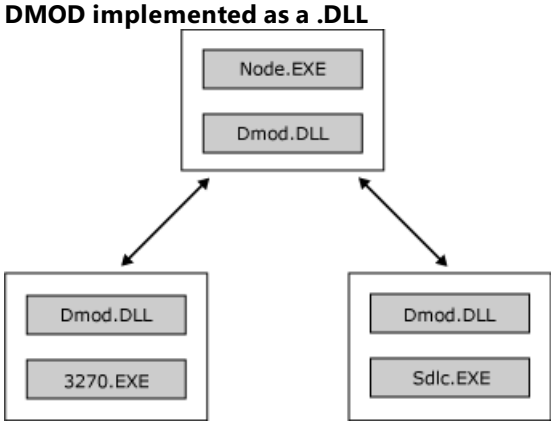
DMOD component providing communications



This figure shows a system consisting of three dynamic localities. Dynamic localities can enter or leave this system at any time.

The DMOD is implemented as a dynamic-link library (DLL). The preceding figure can therefore be represented as follows.

DMOD implemented as a .DLL



Application Localities

Applications such as 3270 emulators can enter dynamically into an SNA server system. The application, in conjunction with the Base, acts as a whole locality and communicates with the other localities in the system using a Dynamic Access Module (DMOD).

The topic DL-BASE/DMOD Interface describes the interface to the Base and the DMOD that enables an application to participate in an SNA server system.

Partners

For Host Integration Server components and applications to communicate with each other, it must be possible to identify a partner within a locality. A partner is an addressable component of a locality; that is, code to which messages can be sent. In a Host Integration Server system, there is generally only one partner within a locality (such as a link service or the 3270 emulation program). However, separate functions within the local 2.1 node (such as the 3270 and APPC functions) can be considered to be separate partners.

Messages

Messages are used to pass data between partners in Microsoft Host Integration Server 2009. This section provides information about message formats.

This section contains:

- Overview of Message Formats
- Buffer Header Format
- Buffer Element Format

Overview of Message Formats

A message always contains fixed-format header information such as a message type and addressing information. It can also contain other header information specific to a particular message type (such as the message subtype) and an indefinite amount of extra data.

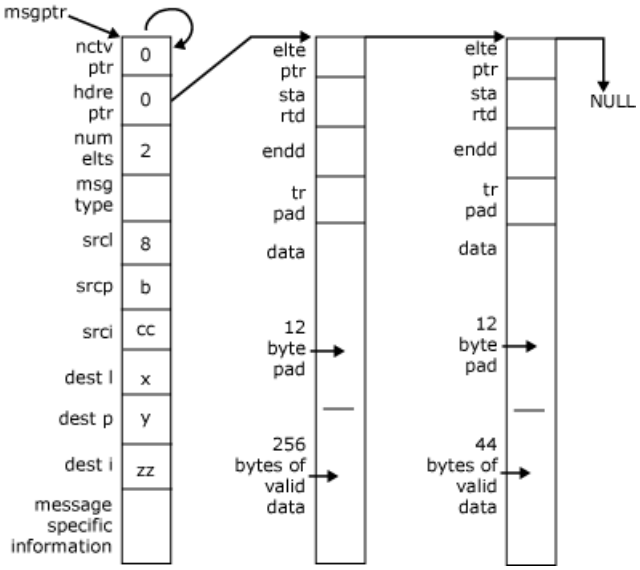
Messages are saved in buffers that consist of one header and zero or more elements:

- The header contains the fixed-format information and a pointer to an element. (This pointer is NULL if there are no elements associated with the message.)
- An element contains any extra data for a message and a pointer to another element if the data continues into another element.

Buffer headers and elements are regarded as contiguous (8-bit) byte sequences. Messages of any length can be built by chaining sufficient elements to a header.

The following figure shows a typical message with two elements.

Typical message with two elements



Buffer Header Format

The following table lists the common fields that always occur at the start of a buffer header. These are followed by further fields specific to the particular message. For details about individual message formats, see FMI Message Formats.

Field	Type	Description
nxtqptr	PTRBFHDR	When the buffer is in a queue, this field points to the header of the next buffer in the queue (NULL if it is the last buffer in the queue). When the buffer is not in a queue, this field points to itself. The SNA server buffer management routines use this field to check for buffer corruption.
hdreptr	PTRBFELT	Pointer to the first buffer element in the associated chain of buffer elements. NULL if the message consists only of a buffer header.
numelts	CHAR	Number of buffer elements chained from the header. Zero if the message consists only of a buffer header.
msgtype	CHAR	Message type. For details, see individual message descriptions in FMI Message Formats.
srcl	CHAR	Source locality. For details, see LPI Addresses.
srcp	CHAR	Source partner. For details, see LPI Addresses.
srci	INTEGER	Source index. For details, see LPI Addresses.
destl	CHAR	Destination locality. For details, see LPI Addresses.
destp	CHAR	Destination partner. For details, see LPI Addresses.
desti	INTEGER	Destination index. For details, see LPI Addresses.

Note

Fields that occupy two bytes, such as **opresid** in Open(PLU) Request are normally represented with the arithmetically most significant byte in the lowest byte address, irrespective of the normal orientation used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. However, the following fields are exceptions:

- The **srci** and **desti** fields in buffer headers are stored in the local format of the application that assigns them (only the assigning application needs to interpret these values).
- The **startd** and **endd** fields in elements are always stored in low-byte, high-byte orientation (the normal orientation of an Intel processor).

Buffer Element Format

The following table lists the common fields that always occur at the start of a buffer element. The **dataru** field contains information specific to the particular message. For details about individual message formats, see FMI Message Formats.

Field	Type	Description
hdreptr->elteptr	PTRBFELT	Pointer to next buffer element in the chain. NULL if this element is the last or only element in the chain.
hdreptr->startd	INTEGER	Start of valid data in this element. The index into **dataru** of the first byte of valid data.
hdreptr->endd	INTEGER	End of valid data in this element. The index into **dataru** of the last byte of valid data.
hdreptr->trpad	CHAR	Pad byte (reserved).
hdreptr->dataru	CHAR[268]	An array of characters that contains the data for this element. Note that the valid data might not occupy the whole of the element. The **startd** and **endd** fields give the indexes into this array of the start and end of the valid data.

Use the following information to help you interpret the message formats:

- Certain messages are shown as having two elements in the message formats. For example, the Open(PLU) Request has the **CICB** field in the first element and the **BIND RU** in the second element. This indicates that the message consists of two distinct linked element chains. The **elteptr** field in the first element points to the second element.

- Fields that occupy two bytes are represented with the arithmetically most significant byte in the lowest byte address, irrespective of the normal orientation used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. The exceptions to this are the **startd** and **endd** fields in elements, which are always stored in low-byte, high-byte orientation (the normal orientation of an Intel processor).

- The offsets indicated by the **startd** and **endd** fields are expressed in terms of the first byte of **dataru** being offset 1. The first byte of valid data is at **dataru[startd–1]**. For example, if **startd** is 11 and **endd** is 18, **dataru** begins with 10 bytes that are not valid data, followed by 8 bytes of valid data.

- It is possible for an element to arrive with **startd** greater than **endd**. This indicates there is no valid data in **dataru**.

In the sample message format shown in Overview of Message Formats, each element has a **startd** of 13, indicating 12 bytes of padding before the start of the valid data. This leaves room for 256 bytes of data, and so the element data (300 bytes long in this example) requires two elements.

LPI Connections

Partners communicate by passing messages to each other. If two partners want to communicate with each other, a locality partner index (LPI) connection is set up between the two partners. Messages then flow between the partners over this connection. The term LPI connection is explained in LPI Addresses. Note that this is not related to the Microsoft® Host Integration Server concept of a connection between the local node and a remote system.

This section contains:

- Paths and DMODs
- LPI Addresses
- Making Connections

Paths and DMODs

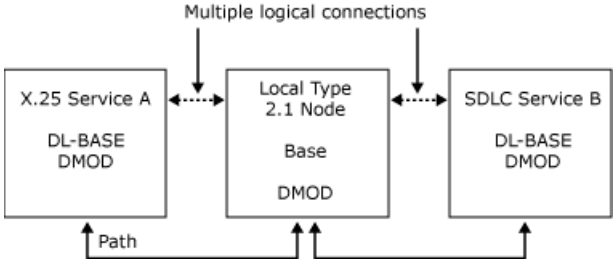
Dynamic Access Modules (DMODs) are responsible for the communication between localities. When the DMODs in two localities can successfully pass messages between them, a path exists between the two localities. A path must exist between two localities before a connection can exist between partners in those localities.

Host Integration Server establishes a path using an appropriate method for the network operating system in use. For example, with Microsoft LAN Manager, a named pipe is used. With NetWare, an SPX connection is used. When the two localities are on the same computer, a local pipe is used. This is implemented using shared buffers to increase performance, but is used by the application in exactly the same way as communication with a remote locality.

The DMOD provides communication between dynamic localities and provides guaranteed in-order delivery of messages flowing over paths between localities. If the DMOD loses its path to another locality, it informs the Base.

The following figure illustrates the paths and connections between an SNA services local node and two 3270 emulation programs. Program A has two connections to the local node (one for each of two sessions). Program B has one connection to the local node.

Paths and connections between an SNA server local node and two 3270 emulation programs



LPI Addresses

An LPI address is used to identify each end of a connection. It has three components: locality (L), partner (P), and index (I), as described in the following list:

- **Locality** is a 1-byte identifier that uniquely identifies a locality within a system. This locality corresponds to an SNA services component (local node, link service, 3270 emulator, and so on).

- **Partner** is a 1-byte identifier that uniquely identifies a partner within the locality. This is not always used, but can be used to distinguish between parts of a component (for example, the 3270 functions in the local node rather than the Advanced Program-to-Program Communications (APPC) functions).

- **Index** is a 2-byte identifier that uniquely identifies a logical entity within the partner. The meaning and use of this field is defined by the communicating partners. It is used to distinguish multiple connections between the same partners (for example, to identify one of many 3270 sessions between the local node and a particular 3270 emulator). The value of zero should not be used as an index value. Applications must assign unique index values for every active LPI connection with the node.

A message flowing over a connection carries a pair of LPIs that identify the source and destination of the message. These are the source LPI and destination LPI of the message. Together they identify the connection on which the message is flowing.

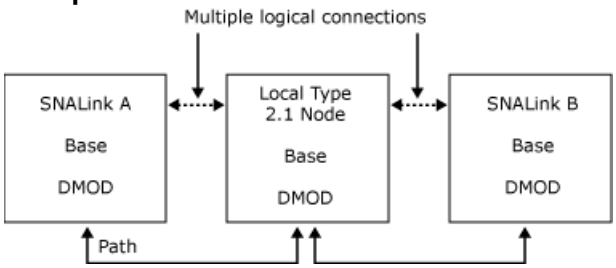
More than one connection can exist between any pair of partners. The I values are then used to distinguish the connections. For example, in communications between the local node and a 3270 emulator, the L and P values identify the message as being 3270 data for that local node, and the I value indicates which session the data is intended for.

The LPIs are assigned by a combination of the partners and the DMODs when the connection is opened, as described in Making Connections.

Because they are assigned dynamically for each component, the L values are not the same across an entire system. For example, a local 2.1 node locality could be known as locality 4 to one 3270 locality, and locality 6 to a second 3270 locality. However, from the viewpoint of any locality, there exists a unique L value for each remote locality within which a path exists. This L value is used as an index into an internal table that identifies the path to that locality.

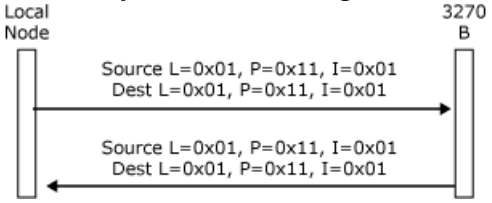
The following figures show an example of the L values that could be used between the components shown in Paths and DMODs, and examples of the LPI values that would be used by the local node on messages flowing between the components. (For more information about how the LPI values are assigned and used, see Opening the PLU Connection.)

Example L values



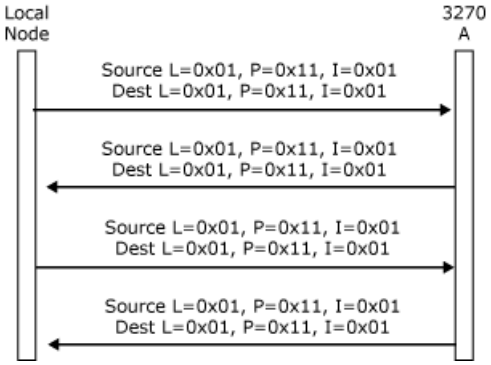
Example L values

L values specified on messages between the local node and 3270 B



L values specified on messages between the local node and 3270 B

LPI values specified on messages flowing on two different connections between the local node and 3270 A



LPI values specified on messages flowing on two different connections between the local node and 3270 A

The Base is called by any piece of code that wants to send a message. It uses the destination L value on the message to determine where to send it. When the message reaches the remote locality, the Base in that locality routes it to the appropriate partner if the locality contains more than one partner.

Making Connections

Before messages can flow across a connection, the connection must be established or opened. This is necessary because a partner (P1) does not initially know the locality partner index (LPI) address of the partner with which it wants to communicate. There may not even be a suitable partner with which to communicate.

A component of the Base, known as the Resource Locator, and a message with a type of Open, known as an Open message, are used to establish a connection between partners.

The following procedure outlines how a connection is established. More specific information is available in Function Management Interface.

To establish a connection between partners

1. The Open message has two forms: an Open request and an Open response. The Open request contains information on the type of partner P1 is looking for.

   P1 fills in an Open request and calls the Base with it. Because it does not know the LPI address of its partner, it sets the destination LPI values to zero.

2. The Base cannot forward the Open to a particular partner, because it has no destination LPI address. Therefore it passes the Open to the Resource Locator, which attempts to find a locality that will accept the Open. The Dynamic Access Module (DMOD) has a record of all the localities that could accept this type of Open. The Resource Locator tries each of these localities until the Open is accepted. If no locality is found, the Resource Locator returns a negative response to the Open to inform the sender that no partner could be found.

3. When a remote locality receives an Open, the Base passes the Open to the partner (P2). If P2 can accept the Open, it responds by sending a positive Open response message to P1.

4. The Open response message returned to P1 contains both the source and destination LPI values for the particular connection. At the end of this exchange, both P1 and P2 know each other's addresses and can communicate over the connection.

The terms source and destination in the context of LPIs refer to the source and destination of the particular message. When the 3270 emulator builds a message to send to the local 2.1 node, it needs to swap the source and destination LPIs received on the Open response from the local 2.1 node.

For a detailed example of how LPI addresses are assigned during initialization of the system services control point (SSCP) and primary logical unit (PLU) sessions, see Opening the PLU Connection.

DL-BASE/DMOD Interface

This section describes the interface to the Host Integration Server 2009 DL-BASE. It includes a listing of the entry points that an application such as a 3270 emulator can call. These entry points allow messages to be sent to and received from services such as the local 2.1 node.

In This Section

DL-BASE/DMOD

The following topics describe an example in which a 3270 emulator is to be adapted to use Host Integration Server 2009. The emulator must communicate with the local 2.1 node.

In This Section

Initialization

The 3270 emulator should initialize the DL-BASE and then call the Dynamic Access Module (DMOD) to obtain the necessary configuration information. This also registers the user name with the DMOD. It can then obtain further system information such as the Host Integration Server version number, if required.

The following table lists the functions involved with this process.

Function	Description
sbpuinit	Initializes the DL-BASE.
sepdcrec	Gets configuration information.
sepdgetinfo	Gets system information.

The **sbpuinit** entry point should always be called before any other DL-BASE/DMOD entry points except SNAGetVersion. For new emulators, **sepdcrec** should be called after **sbpuinit**. (Because of the order of calls used in older emulators, a call to **sepdcrec** before **sbpuinit** is still supported, but this order is not recommended.)

Sending Messages

The 3270 emulator inserts a message in a buffer and then calls the DL-BASE to send it. The message contains source and destination locality partner indexes (LPIs), which are set up when the connection is opened. For more information, see LPI Connections.

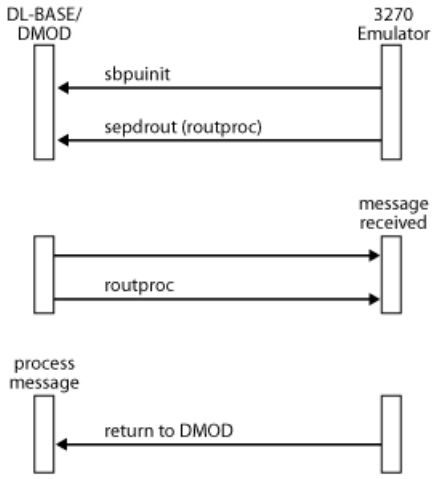
The application can either obtain a new buffer to contain the message to be sent (using sepdbubl), or reuse a buffer in which it previously received a message. The application is responsible for any buffer it has obtained or in which it has received a message. The application must either use (or reuse) the buffer to send a message or release it (using sepdburl). If the buffer to be reused does not contain the correct number of elements for the message to be sent, the application can obtain additional elements (using sbpibegt) or release existing ones (using sbpiberl). In this case, it must also ensure that the **numelts** field in the buffer header indicates the correct number of elements.

The function used to send the message is sbpusend.

Receiving Messages

The following figure shows the method for receiving messages from the Dynamic Access Module (DMOD).

Receiving messages using a routing procedure



After DMOD initialization, the 3270 emulator registers the routing procedure by calling sepdrout. When the DMOD receives a message, it calls the 3270 emulator routing procedure, which can then process the message.

With this approach, there is no context switch between the DMOD thread and the 3270 emulator thread. However, the routing procedure must return control to the DMOD quickly. For instance, it cannot suspend waiting for a keyboard input.

The application must determine whether the received message is for this application or for another application. If the message is not for this application, the routing procedure must return, indicating that the message was not processed. If the application processes the message, it is responsible for freeing the buffer when the processing is finished.

In some cases, the routing procedure can process the message to completion. An alternative is for the routing procedure to put the message on an application queue and then clear an application semaphore. The application can then subsequently process the message.

Performance can be improved further by sending a Status-Resource message (to return credit to the local node, enabling it to send further data) from the routing procedure when a message is received, rather than waiting until the message is processed to completion. This usage is illustrated in Sample Code: Initialization and Routing Procedure. For more information about credit and flow control, see Pacing and Chunking.

After the application has received a message, the application is responsible for the buffer in which the message was received. The application must either reuse the buffer to send a message (using sbpusend) or release it (using sepdburl). If the buffer to be reused does not contain the correct number of elements for the message to be sent, the application can obtain additional elements (using sbpibegt) or release existing ones (using sbpiberl). In this case, the application must also ensure that the **numelts** field in the buffer header indicates the correct number of elements.

Opening a Connection

Before a locality partner index (LPI) connection can be used to transfer data, it needs to be opened. This is performed by sending Open messages, starting with an Open request. The format of an Open message is defined by the interface being used. For example, the 3270 emulator uses the function management interface (FMI) to communicate with the local 2.1 node.

The interface also defines the initiator of the Open request. In this case, the 3270 emulator sends the Open(SSCP) Request, and the local 2.1 node sends the Open(PLU) Request.

On the **Open(SSCP) Request**, the 3270 emulator sets all the source and destination LPIs to zero, except for the source index, which can be used by the 3270 emulator for internal routing (for example, to distinguish between two sessions).

The DL-BASE and Dynamic Access Module (DMOD) ensure that Open messages are routed to a suitable destination. If a routing procedure is used, it should always first call sbpurcvx to process Open responses. When **sbpurcvx** indicates that it has not processed a received message, and the received message is an Open OK response, the application is informed that the connection was established successfully.

Termination

The 3270 emulator must call sbputerm to free DL-BASE/Dynamic Access Module (DMOD) resources before it terminates.

DL-BASE/DMOD Entry Point Summary

The following table shows entry points divided into the categories DL-BASE, Dynamic Access Module (DMOD), and buffer management, and listed in alphabetical order within each category.

DL-BASE entry points

DL-BASE entry points	Description
sbpuinit	Initializes the DL-BASE.
sbpurcvx	Processes Open from routing procedure.
sbpusend	Sends message.
sbputerm	Terminates.

DMOD entry points

DMOD entry points	Description
routproc	Sample routing procedure.
sepdchnk	Gets the function management interface (FMI) chunk size.
sepdcrec	Gets user and diagnostics records from configuration file.
sepdgetinfo	Gets SNA server system information.
sepdrout	Sets up the routing procedure (Microsoft® Windows Server™ 2003 and Windows® 2000 Server only).

Buffer management entry points

Buffer management entry points	Description
sbpibegt	Gets the buffer element.
sbpiberl	Releases the buffer element.
sepdbubl	Gets the buffer.
sepdburl	Releases the buffer.

Note

The standard-call convention (CDECL) is used on Windows Server2003 and Windows2000.

Note

The format of buffer headers and elements is described in Messages. The formats of individual messages contained in buffer s are defined in FMI Message Formats.

Sample Code: Initialization and Routing Procedure

This topic contains an outline of source code for receiving messages from the Dynamic Access Module (DMOD).

Note

TRACE*n*() is a macro used to specify data to be traced. This data can include variable parameters. The value **n** identifies the severity level of the trace. The unmatched parentheses are deliberate. They are resolved by the expansion of the macro.

```
/*****************************************************************/
/* Sample code for initialization and routing procedure.         */
/*****************************************************************/

HSEM  dummysem = NULL;      /* This semaphore is never used      */

/*****************************************************************/
/* Initialization procedure                                      */
/*****************************************************************/
USHORT init_proc()
{
  COM_ENTRY("initp");
  rc = sbpuinit(&dmodsem, CLIENT, CES3270, username);
  TRACE4()"DMOD initialized, rc=%d",rc));
  if (rc == NO_ERROR)
  {

    /*****************************************************************/
/* The procedure routproc will be called whenever a message is   */
/* received by the DMOD. This is used to post back the application, */
/* but take care to protect any queues against concurrent access by */
/* multiple threads.                                             */
/*****************************************************************/
    rc = sepdrout(routproc);
    TRACE4()"Rout proc set up, rc=%d",rc));
    if (rc == NO_ERROR)
    {
      /* Other initialization here                               */
    }
  }
  return (rc);
}
/*****************************************************************/
/* The routine routproc is called whenever the DMOD receives a   */
/* message or a status indication.                                */
/*****************************************************************/
USHORT FAR _loadds routproc(buf, srcl, status)
BUFHDR FAR *buf;        /* Buffer that has been received         */
USHORT      srcl;       /* Locality from which buffer was received  */
USHORT      status;     /* Reason for call                       */
                        /* CEDINMSG = message received           */
                        /* CEDINLLN = path error occurred (on srcl) */
{
  COM_ENTRY("routp");   /* initialize rc=FALSE                   */

                        /* Call the DL BASE to handle re-resource   */
                        /* location                              */
  if (!sbpurcvx(&buf, srcl, status))
  {
    switch (status) {
      case CEDINMSG:
        if (buf->destp == S3PROD)         /* Is the message for us? */
        {
          /*****************************************************************/
     /* Process the received message.                            */
     /*                                                          */
```

```
      /* If the message is DATAFMI on the PLU-SLU session, and the   */
      /* application has requested to use flow control on the        */
      /* session, then this processing should include:               */
      /*                                                             */
      /* -  increment number of messages received by the client      */
      /* -  check whether the number received exceeds the threshold  */
      /*    for normally returning credit to the node. If so, check  */
      /*    whether it is OK to return credit (for example, not short of      */
      /*    buffers), and if OK send a status-resource message to    */
      /*    the node to give it credit to send more messages to the  */
      /*    client.                                                  */
  /*************************************************************/
           rc = TRUE;
           TRACE2()"Routing proc got message at %p",buf));
         }
         else
         {
           TRACE2()"Routing proc did not take message at %p",buf));
         }
         break;

       case CEDINLLN:
         TRACE2()"Path error on %d",srcl));
             /*************************************************************/
         /* Process the path error status.                          */
  /*************************************************************/
         break;
     }

     /*************************************************************/
     /* If the message/status cannot be completely processed here,   */
     /* the application can queue the message and clear a semaphore for the  */
     /* main thread to continue the processing.                     */
  /*************************************************************/
   } else {
     rc = TRUE;       /* DLBase handled the message on our behalf    */
   }
   /* Returning a value of TRUE indicates that we processed the      */
   /* event return(rc);
  }
```

Function Management Interface

The function management interface (FMI) provides applications with direct access to SNA data flows and information about SNA control flows by means of status messages. This section provides information about the SNA sessions and connections over which FMI messages can flow, and summarizes the messages. The FMI is particularly suited to the requirements of 3270 emulation applications.

In This Section

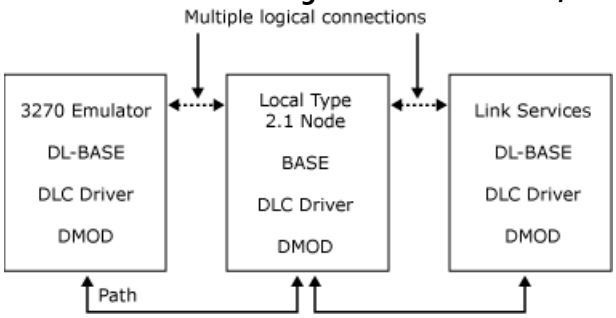
FMI Concepts

The local node provides the SNA layers of path control, transmission control, and data flow control (DFC), as well as logical unit (LU) services as shown in the following figure. In terms of the SNA layers, the function management interface (FMI) is between presentation services and DFC. This means that most of the SNA protocol handling is performed by the local node. In particular, the DFC layer of the local node is responsible for the state changes associated with chaining, bracket, and quiesce protocols.

The FMI is defined in terms of the messages that are sent across the interface. Note that this is logically distinct from the definition of the DL-BASE/Dynamic Access Module (DMOD) interface, which defines the mechanism for sending messages between two components in Microsoft® Host Integration Server (for example, between the local node and the 3270 emulator).

The FMI is used by LU types 0, 1, 2, and 3, but not by LU type 6.2. It provides access to the system services control point (SSCP)-LU session as well as the main primary logical unit (PLU)-SLU session. (For more information about these sessions, see Sessions and Connections.) An application can use the FMI to access multiple sessions and hence multiple LUs, simultaneously.

Emulator communicating with the local node, which communicates with the link service



In this example, the 3270 emulator on the client communicates over the local area network (LAN) with the local node on the server by exchanging messages. The content and format of the messages are defined by the FMI. The DMOD is used to transport the messages, but does not interpret them. The local node provides the SNA service for formatting the message. The link service and the data link control (DLC) driver are responsible for transferring data between the local node and the DLC adapter.

In This Section

- Sessions and Connections
- Application Flags

Sessions and Connections

An application using the function management interface (FMI) can communicate with the host on two SNA sessions as described in the following list:

- The system services control point (SSCP) session, between an SNA server logical unit (LU) and the host SSCP, provides information about the activation of the LU and supports communication with the SSCP for commands such as character-coded and field-formatted logon and logoff commands. There is one SSCP session for each SNA server LU.

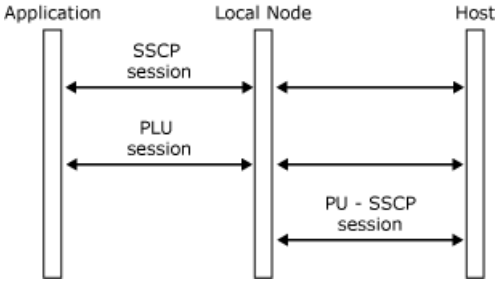
- The primary logical unit (PLU) session, between an SNA server LU and the host PLU, is the main session for data transfer between the local application and the host application. There is one PLU session for each SNA server LU.

The local node communicates directly with the host on the physical unit (PU)-SSCP session:

- The PU-SSCP session, between the PU (local node) and the host SSCP, supports the reporting of alert information and link statistics to the host SSCP.

The following figure shows the three sessions.

Three sessions



The application can communicate with the local node by means of two locality, partner, index (LPI) sessions. Rather than specifying the session on which a message is to flow, the application sends the message to the local node on one of these connections. The local node then routes it to the appropriate SNA session.

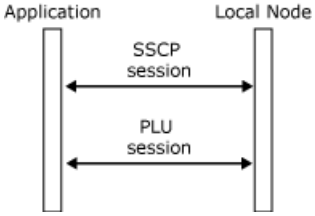
The connections are used as follows:

- The SSCP session is used for the initial startup and logon information for a 3270 session. The Host Integration Server 3270 emulation programs also send network management information, such as user alerts and Response Time Monitor (RTM) statistics, to the local node on this session. For more information about this connection, see SSCP Connection.

- The PLU session is used for the transfer of application data, and for status and flow control messages between the application and the local node. For more information about this connection, see PLU Connection.

The following figure shows these sessions.

Connections between an Application and a local node



These sessions are specific to the local node and the application. Data and status messages passed across a connection result in SNA data and SNA control requests being sent on the appropriate SNA session. Similarly, SNA data and SNA control responses received on an SNA session result in data and control messages being passed to the application on the appropriate connection.

The relationship between the three SNA sessions and the two connections is as follows:

- SNA messages on the SSCP session from the host to the local node result in messages from the local node to the application on the SSCP connection. Messages from the application to the local node on the SSCP connection normally

result in SNA messages on the SSCP session from the local node to the host (with the exception of network management information, which results in messages on the PU-SSCP session).

- SNA messages on the PLU session from the host to the local node result in messages from the local node to the application on the PLU connection. Messages from the application to the local node on the PLU connection result in SNA messages on the PLU session from the local node to the host.

- SNA messages on the PU-SSCP session from the local node to the host are generated by messages from the application to the local node on the SSCP connection. When the application sends network management information such as 3270 user alerts on the SSCP connection, the local node distinguishes it from other data on this connection (which normally corresponds to the SSCP session) and sends the appropriate information about the PU-SSCP session to the host. For more information, see 3270 User Alerts.

Note the distinction between these SNA sessions and 3270 emulation sessions. A 3270 emulator can have more than one 3270 emulation session. For each emulation session, there are separate SSCP and PLU sessions.

Each connection between the application and the local node is opened, managed, and closed separately. This means that an application must maintain a separate internal control block containing the LPI pair, message keys, and state of the connection for each of the SNA sessions associated with each 3270 emulation session. For example, an application using three 3270 emulation sessions, each with an SSCP session and a PLU session, will require six control blocks.

An application identifies the connection (and hence the session) to which a particular message belongs using the LPI pair present in the message. On a received message, the destination index (I) value contains the application's identifier for the connection, and the source I value contains the local node's identifier for the connection. These are reversed for messages sent by the application.

The application selects the LU within the local node that it can use for communications by the relationship in the configuration table between the LU record and APPL record. (For more information, see Opening the SSCP Connection.) The application may be unaware of which LU it accesses if the LUs are arranged within LU groups.

See Also

Reference

Application Flags

Application Flags

Application flags are included on the following messages:

- All Data messages (both inbound and outbound)
- Status-Acknowledge(Ack) (outbound only)
- Status-Acknowledge(Nack-1) (outbound only)
- All Status-Control messages (both inbound and outbound)

These flags represent key indicators of the state of the session to which the message relates and are closely related (but not always equivalent) to the request header or response header (RH) indicators in the SNA request or response. Note that for inbound messages, applications need to set the flags on **Data** messages and **Status-Control** messages only.

For outbound messages, the local node sets the application flags to reflect the contents of the RH in the corresponding SNA message. The local node performs checks on the SNA message before sending it to the application. Therefore, the application can assume that the RH indicators follow the SNA protocols and need not perform its own checks. The application's task in interpreting the application flags is much simpler than if the local node presented the message with the uninterpreted RH. For example:

- If the application specified the segment delivery option when the primary logical unit (PLU) connection was opened, the end chain indicator (ECI) on an SNA request will occur on the first segment of the last request/response unit (RU) in a chain, but the chain is not complete until the last segment of that RU is received. In this case, the local node manipulates the application flags so that the ECI flag is set in the last segment rather than the first. (For more information, see Opening the PLU Connection.)
- Applications using Transmission Service profile 4 (TS profile 4) on the PLU session can receive the definite response 2 (DR2) RH indicator in combination with definite response 1 (DR1) or exception response (ER) to give RQD2, RQD3, RQE2, and RQE3 requests. The local node interprets the RH indicators and sets the **COMMIT** application flag accordingly.

For inbound Data and Status-Control messages, you should set the application flags to control session characteristics such as chaining, direction control, and brackets. For **Status-Acknowledge** messages, the local node generates an SNA response and sets the RH indicators using information saved from the corresponding request. The application does not need to set the flags on this message.

For information about application flag usage when you are using function management interface (FMI) chunking, see Chunking.

In most cases, the application does not need to use the application flags on Status-Acknowledge(Ack) messages, which derive from the response header indicators on the corresponding response. However, certain applications do require access to the response header flags on responses. For example, transaction-processing applications using TS profile 4 can receive the DR2 flag on responses, which appear as the **COMMIT** flag in the application flags.

Application flag usage on Status-Control (SC) messages is derived from the response header indicators in the corresponding data flow control or session control request unit. Applications may need to be aware of the response header flags for Status-Control messages. For example, LUSTAT request type 6 is a no-op used solely to enable response header flags to be sent when no other request is allowed. The local node delivers the request to the application as a Status-Control(LUSTAT) Request with the relevant application flags set. For summaries of valid request header usage for data flow control request units and of valid response header indicators for SC requests, see *SNA Format and Protocol Reference Manual: Architectural Logic* (IBM publication SC30-3112).

In the summary of the application flags in the table that follows, bits are numbered with bit 0 as the most significant bit in a byte and bit 7 as the least significant. An application flag is set if the relevant bit for the flag is 1 and not set if the bit is 0.

Flag 1 occurs in all messages.

The following table lists the meanings of the individual bits.

Bits in flag 1	Meaning
FMHI [bit 0, flag 1] Value: AF_FMH (0x80)	Function management header indicator. Set if a function management header is present in the message, or if the message is a function management data network services (FMD NS) request. Only valid on Data messages. This flag is always set for 3270 user alerts, which are sent on the system services control point (SSCP) connection. For more information, see 3270 User Alerts.
BCI [bit 1, flag 1] Value: AF_BC (0x40)	Begin chain indicator. Set if this message starts a chain. For more information, see Outbound Chaining and Inbound Chaining.
ECI [bit 2, flag 1] Value: AF_EC (0x20)	End chain indicator. Set if this message ends a chain. For more information, see Outbound Chaining and Inbound Chaining.
COMMIT [bit 3, flag 1] Value: AF_COMM (0x10)	Commit indicator. Set if chain carries DR2.
BBI [bit 4, flag 1] Value: AF_BB (0x08)	Begin bracket indicator. Set if chain carries begin bracket (BB). Note that this does not necessarily indicate that the bracket has been initiated. For more information, see Brackets.
EBI [bit 5, flag 1] Value: AF_EB (0x04)	End bracket indicator—set if chain carries end bracket (EB). Note that this does not indicate that the bracket has terminated. For more information, see Brackets .
CDI [bit 6, flag 1] Value: AF_CD (0x02)	Change direction indicator. Set if chain carries change direction (CD). For more information, see Direction.
SDI [bit 7, flag 1] Value: AF_SD (0x01)	System detected error indicator. Set if the local node detects an error in outbound data. For more information, see Outbound Data.

Flag 2 occurs in all messages except **Status-Control(STSN)**, where the indicators included in this byte are not applicable.

The meanings of the individual bits are listed in the following table.

Bits in flag 2	Meaning
CODE [bit 0, flag 2] Value: AF_CODE (0x80)	Alternate code indicator. Set if the alternate code set (usually ASCII) is used for this Data message. Note that function management headers are unaffected by the code selection indicator.
ENCRYP [bit 1, flag 2] Value: AF_ENCR (0x40)	Enciphered data indicator. Set to indicate that the information in the **Data** message is enciphered under session level cryptography protocols. You must provide the necessary support for data encryption. The Host Integration Server local node does not support cryptography.
ENPAD [bit 2, flag 2] Value: AF_ENPD (0x20)	Padded data indicator. Set in conjunction with the **ENCRYP** flag to indicate that the data was padded at the end to the next integral multiple of eight bytes before enciphering.
QRI [bit 3, flag 2] Value: AF_QRI (0x10)	Queued response indicator. Set if the response to this request is to be queued in the transmission control and data flow control layers. This flag is only significant for inbound messages.
CEI [bit 4, flag 2] Value: AF_CEI (0x08)	Chain ending indicator. Set on a message corresponding to an outbound SNA request with EC and begin basic information unit (BBIU). This flag is provided solely for the use of SNA server components. Your application should not attempt to use it.

BBIUI [bit 5, flag 2] Value: AF_BBIU (0x04)	Begin basic information unit indicator. Set on a message corresponding to an outbound SNA request with BBIU. This flag is provided for the use of SNA server components and for applications using segment delivery and outbound pacing together. Your application should not attempt to use it. (For more information, see Pacing and Chunking.)
EBIUI [bit 6, flag 2] Value: AF_EBIU (0x02)	End basic information unit indicator. Set on a message corresponding to an outbound SNA request with end basic information unit (EBIU). This flag is provided solely for the use of SNA server components. Your application should not attempt to use it.
RBI [bit 7, flag 2] Value: AF_RBI (0x01)	Real BID indicator. Set on **Status-Control(BID) Request** messages from the local node only. 0x01 indicates that the message is due to an SNA BID RU. 0x00 indicates that the message is due to an outbound function management data (FMD) RU with BB set.

See Also

Concepts

Sessions and Connections

SSCP Connection

The system services control point (SSCP) connection of the application to the local node provides access to the SSCP session between the Microsoft® Host Integration Server secondary logical unit (LU) and the host SSCP.

For simplicity, this section describes the SSCP connection as if an application only uses a single SNA server LU (and therefore a single SSCP connection). In practice, applications can use multiple LUs.

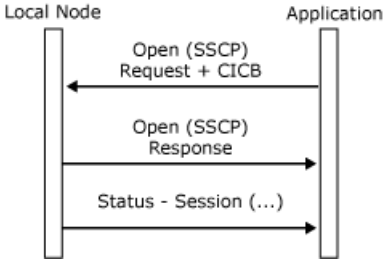
In This Section

Opening the SSCP Connection

An application gains access to the system services control point (SSCP) session by opening an SSCP connection to the local node. To do this, an application sends an Open(SSCP) Request message to the local node, which responds with an Open(SSCP) Response. The local node follows a positive **Open(SSCP) Response** with a Status-Session message reporting the current state of the SSCP session. (For more information, see Using the SSCP Session.)

The following figure shows the message flow. For a figure showing a more detailed message flow, including locality, partner, index (LPI) values used during initialization of both the SSCP and primary logical unit (PLU) sessions, see Opening the PLU Connection.

Message flow between a local node and an application



In This Section

- LU Groups
- Resource Location for Open SSCP

LU Groups

Note that Host Integration Server supports logical unit (LU) groups. An LU group consists of a number of LUs of the same type, such as 3270 display LUs or logical unit application (LUA) LUs. Any of the LUs in the group can be used for the same task.

If an application sends an Open(SSCP) Request specifying the name of a 3270 display LU group, the local node can select any LU within the group to be used by the application. LUA LU groups are used in the same way, except that the application can specify either the name of the group or the name of any LU within the group to access the group.

See Also

Concepts

Resource Location for Open SSCP

Resource Location for Open SSCP

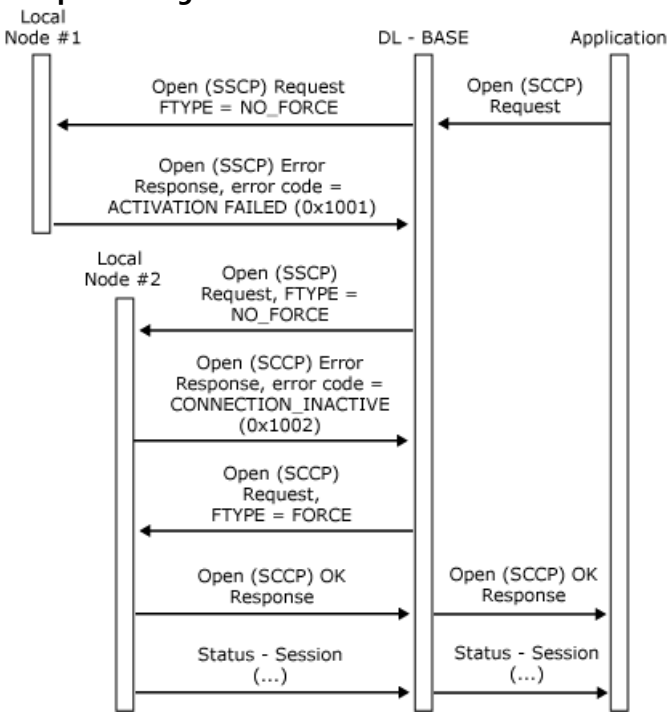
When attempting to find a free logical unit (LU) across more than one local node, the application does not need to know which local node owns the LU. The DL-BASE is responsible for finding a suitable local node, using the mechanism described. The description is intended to assist in interpreting traces of the message flows involved, and is not needed to write an application

The open force type field in the Open(SSCP) Request specifies either a forced or nonforced Open. If the LU for which the Open is intended does not have an active system services control point (SSCP) session because its link is inactive, a forced Open instructs the local node to attempt to activate the link and the SSCP session. A nonforced Open succeeds only if the SSCP session is already active, and otherwise returns with an error code indicating the state of the LU's connection.

When the application issues the **Open(SSCP) Request**, it does not set the open force type field. The DL-BASE issues a nonforced Open to each node in turn until it finds an LU that already has an active SSCP session. If none of these Opens succeeds, the DL-BASE issues a forced Open to the node that returned the best error code—that is, the one most likely to be able to activate the session.

The sample message flows in the following figure show this process for two local nodes. The DL-BASE tries each in turn, using nonforced Opens. The error code from node #2 indicates that it is more likely to be able to activate the SSCP session than node #1, so the DL-BASE sends a forced Open to node #2. The application is aware only of the first request and its response.

Sample message flow for two local nodes



To enable applications to restart after a disastrous failure (such as terminating the 3270 emulation program), the local node also accepts an Open(SSCP) Request from an application that has failed and has been restarted, providing the same source locality, partner, index (LPI) fields are used. In this case, a **TERM-SELF** message is sent to the host if the LU is bound.

The SNA server LU through which the application communicates is selected by the relationship between the APPL record and the LU or LU group record in the configuration file. The application specifies its name using the source name field on the **Open(SSCP) Request**. The local node fills in the LU or LU group number, selects an unused LU within the LU group (if the association is to an LU group), and informs the application of this LU number on the Open(SSCP) Response.

The **Open(SSCP) Request** specifies the following:

- The source application name.
- A resource identifier that can be used by the application to correlate the Open(PLU) Request that is sent to the application. (For more information, see Opening the PLU Connection.)
- A connection information control block, which specifies the response header usage, checks that the local node should perform for the LU. If the field for a code is set to 0x01, that receive check will be carried out by the data flow control layer of the local node on data arriving from the host. The corresponding send checks are unaffected and are always performed. The connection information control block is provided because these receive checks are optional in SNA.

However, it is anticipated that most applications will require all these checks to be performed (all values set to 0x01).

- An indicator that specifies whether the application is to be treated as high or low priority. All SNA server 3270 LUs are marked as high priority (printers do not send significant data inbound). The effect of high priority is to enable data to be progressed faster to the host when the link is busy.

- An indicator that specifies whether the application is an LUA. This determines whether the local node and the application will communicate using the LUA variant of the function management interface (FMI). (For more information, see FMI Concepts.)

- An indicator that specifies a nonforced or forced Open. This determines whether the local node will attempt to activate the SSCP session if it is not currently active.

The Open(SSCP) Request can fail for one of several reasons, which can be determined from the error codes on the Open(SSCP) Response sent to the application, as detailed in the following list:

- The local node may still be initializing (retrieving information from the configuration file). In this case, the application can retry immediately.

- The configuration file may not have an entry for the application, or the application record in the configuration file may not point to an LU or LU group record.

- For a nonforced Open, the SSCP session may be inactive.

See Also
Reference
Resource Location for Open SSCP

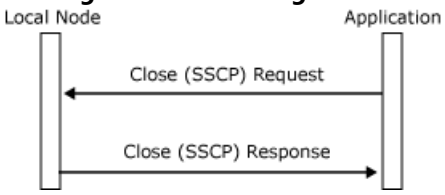
Closing the SSCP Connection

To close the system services control point (SSCP) connection, an application sends a Close(SSCP) Request to the local node, which responds with a Close(SSCP) Response. The **Close(SSCP) Request** is unconditional. The **Close(SSCP) Response** always reports that the connection was successfully closed. The **Close(SSCP) Response** is provided so that applications can determine when outstanding data and status messages on the session have been delivered.

If the logical unit (LU) is bound, the local node sends a **TERM-SELF** message to the SSCP on behalf of the application to elicit an **UNBIND**. An application that needs to be unbound can issue **Close(PLU)**. (For more information, see Closing the PLU Connection.) Normally, the SSCP connection can be maintained while the application task is active, even if it is idle.

Closing the connection invalidates the locality, partner, index (LPI) pair for the connection, but does not alter the state of the SSCP session. The following figure shows the message flow.

Message flow for closing a connection



See Also

Concepts

RTM Parameters

3270 User Alerts

Other Resources

Opening the SSCP Connection

Using the SSCP Session

Using the SSCP Session

When the application has opened a system services control point (SSCP) connection, it has access to the SSCP session and can send data to the host SSCP.

In This Section

- SSCP Session Characteristics
- SSCP Session Status

SSCP Session Characteristics

For SNA type 2.1 nodes, the system services control point (SSCP) session uses function management (FM) profile 0 and Transmission Service profile (TS profile) 1. This combination of profiles provides the following session characteristics:

- The primary and secondary half-sessions both use immediate request mode.
- The primary and secondary half-sessions both use immediate response mode.
- Only definite-response single request unit chains are allowed.
- The maximum request unit size is limited to 256 bytes.
- Data flow control request units are not supported.
- Pacing is not supported.
- Identifiers are used (rather than sequence numbers) on the normal flows.

This implies that the SSCP connection has the following characteristics:

- All Data messages have the acknowledgment required (ACKRQD) field set.
- All Data messages have the begin chain indicator (BCI) and end chain indicator (ECI) application flags set.
- Status-Control messages do not flow on the connection.
- Status-Session messages from the local node to the application only report changes in the activation state of the session.
- The chaining, bracket, confirmation, and recovery protocols (described in PLU Connection) do not apply.

Using the SSCP connection, the application can send and receive **Data** messages corresponding to function management data network services (FMD NS) (session services) requests and FMD data requests. Examples of FMD NS (session services) requests are:

- **INIT-SELF.** Requests from the secondary to the host SSCP requesting that the SSCP assist in the initiation of a session to the host PLU, effectively requesting a **BIND**. (For more information, see Opening the PLU Connection.)
- **TERM-SELF.** Requests from the secondary to the host SSCP requesting that the PLU-SLU session be terminated with an **UNBIND**. (For more information, see Closing the PLU Connection.)
- **Character-coded requests.** Requests such as logon, logoff, or test commands from the secondary display, or a logon prompt from the host application.
- **NOTIFY.** Requests used by the secondary to notify the host SSCP that a device is available after a **BIND** was rejected with sense code 0x0845, for example, where a device emulator supports logical power-off.

The local node sends a **NOTIFY** request to the SSCP on behalf of the LU whenever the application's SSCP connection state changes while the LU is active. A **NOTIFY** (vector key 0x0C with byte 5 set to 0x03), which can act as secondary LU, is sent in the following cases:

- When an Open(SSCP) Request is received when the LU is already active.
- When an ACTLU request is accepted when the SSCP connection is already opened.

A **NOTIFY** (vector key 0x0C with byte 5 set to 0x01), which cannot currently act as secondary LU, is sent in the following cases:

- When an ACTLU is received when the SSCP connection is not open.
- When a Close(SSCP) Request is received when the PLU session is not bound.
- When an **UNBIND** request is received when the SSCP connection is not open.
- When the long response including the **NOTIFY** vector is used for ACTLU requests.

These **NOTIFY** messages can be used by the host in conjunction with the negative response 0x0845 that the local node gives to a **BIND** received when the SSCP connection is not open. (For more information, see Opening the PLU Connection.)

SSCP Session Status

While the system services control point (SSCP) connection is open, the local node reports the initial state and any subsequent changes of state of the SSCP session to the application using Status-Session messages. There are four distinct **Status-Session** status codes that can occur for the SSCP connection:

- **No-Session.** The SSCP session between the SNA server logical unit (LU) and the host SSCP is not active because the SNA server physical unit (PU) or LU is not activated. The **Status-Session** carries a qualifying status code to indicate why the SSCP session is inactive. The application cannot use the SSCP connection to send data to the host SSCP. The qualifiers are:

    - **PU-INACTIVE.** Activate PU (ACTPU) has not been received or Deactivate PU (DACTPU) has been received.

    - **PU-ACTIVE.** ACTPU(COLD) has been received from the SSCP.

    - **PU-REACTIVATED.** ACTPU(COLD) has been received while the PU was active. (The application is not informed if ACTPU(ERP) is received while the PU is active.)

    - **LU-INACTIVE.** ACTLU has not been received, or DACTLU has been received.

- **Link-Error.** The SSCP session between the SNA server LU and the host SSCP is not active, due to a data link control (DLC) error. The **Status-Session** carries a qualifying status code that gives the error code reported by the DLC. The application cannot use the SSCP connection to send data to the host SSCP.

  Note that this session state is reported when the local node is informed that the locality containing the Host Integration Server Synchronous Data Link Control (SDLC) link service has been lost due to a path failure. The qualifier 0x0D is used. The link service will close the link when it is informed of the path error so the application can treat this as an outage.

- **LU-Active.** The SSCP session is active due to the receipt of ACTLU. The application can use the SSCP connection to send data to the host SSCP.

- **LU-Reactivated.** The SSCP session has been reactivated due to the receipt of an ACTLU from the host SSCP. The SSCP connection is still active, but data may have been lost.

For more information, see Status-Session Codes.

RTM Parameters

The Status-RTM message is sent to the application by the local node to indicate the Response Time Monitor (RTM) parameters being used by the host. The host can specify the following parameters:

- Whether RTM measurement is active or inactive.
- Whether local display of RTM data by the application is permitted.
- The definition by which response times are to be measured:
    - Until the first character of a response is written to the screen.
    - Until the keyboard is unlocked.
    - Until the application is allowed to send data. Change direction (CD) or end bracket (EB) is received.
- The boundaries by which response times are to be classified into time bands.
- The initial values of the counters, which indicate how many responses have been received in each time band (as defined by the boundaries).

The local node is responsible for interpreting the response times reported to it by the application, and for sending RTM statistics to the host when required. The application is responsible for measuring the response times and reporting them to the local node. (The application reports response times to the local node using the Status-Acknowledge message. For more information about measuring and reporting response times, see Response Time Monitor Data.)

If the application does not need to provide a local display of RTM data, it only needs to determine whether RTM measurement is active. If active, it needs to determine the definition by which response times are measured. It can ignore the other parameters. If RTM measurement is not active, the application need not measure and report response times.

If the application needs to provide a local display of RTM data, it should use the information from the Status-RTM message to ensure that the local interpretation of response times matches the interpretation used by the host. In particular, it should not display RTM data at all if the **Status-RTM** message indicates that local display is not permitted (or if the **permission to view RTM data** field in the 3270 user configuration record indicates that it is not permitted). The application is responsible for maintaining its own RTM statistics for local display, that is, for classifying the response times according to the boundaries specified by the host and maintaining counts of responses in each category. Although the local node maintains these statistics for sending to the host when required, it does not report them to the application.

Note
RTM statistics are maintained for a specific logical unit (LU), not for a specific application's use of that LU. This means that wh en the **Status-RTM** message is received at start of day, the counters can be nonzero to include a previous use of the LU. The counters are only reset when the host requests the local node to reset them or when the local node sends unsolicited RTM da ta to the host.

See Also
Concepts
Closing the SSCP Connection
3270 User Alerts
Other Resources
Opening the SSCP Connection
Using the SSCP Session

3270 User Alerts

The Host Integration Server 3270 emulation program can send 3270 user alerts to the local node on the system services control point (SSCP) connection. This enables the local node to route each alert to the appropriate host for the 3270 session on which it was sent.

To send a 3270 user alert, the application should send it as a Data message on the SSCP connection. The local node will recognize it as a 3270 user alert if both of the following are true:

- The function management header indicator (FMHI) bit in the application flag is set.
- The first three bytes of the data are 0x41038D, indicating a Network Management Vector Transport (NMVT).

The local node sends the alert to the appropriate host for the 3270 session on which it was received. If a relative time subvector is present (0x42) with increment type 0xEF (sequence), the local node sets the sequence number in each message (starting at one from power-up and incrementing by one for each message sent). Host Integration Server 2009 allows sequence number values up to 2^16. Apart from this, the local node does not alter the contents of the alert.

Note
There can be some delay before the application receives a response to the alert. The response is sent on the SSCP connection in the same way as other data on this connection. The application must not send further data on the SSCP connection (including further alerts) until it has received a response to this alert.

See Also
Concepts
Closing the SSCP Connection
RTM Parameters
Other Resources
Opening the SSCP Connection
Using the SSCP Session

PLU Connection

The primary logical unit (PLU) connection of the application to the local node provides access to the PLU session between the Microsoft® Host Integration Server's logical unit (LU) and a PLU in the host.

This section describes how an application opens and closes its PLU connection, and the use of the PLU connection.

For simplicity, this section describes the PLU connection as if an application uses only a single SNA server LU (and therefore a single PLU connection). In practice, applications can use multiple LUs.

In This Section

Opening the PLU Connection

The opening of the primary logical unit (PLU) connection is closely associated with the establishment of the PLU session. The local node opens the PLU connection when it receives a **BIND** command from the host for a logical unit (LU) for which an application has previously opened a system services control point (SSCP) connection. Possible sequences are:

- An application opens its SSCP connection and sends a character-coded logon request or **INIT-SELF** request to the host SSCP. A host PLU subsequently sends a **BIND** request to the SNA server LU, and the local node opens the PLU connection.

- A host PLU sends an unsolicited **BIND** command to the SNA server LU. If the SSCP connection for the LU is open, the local node opens the PLU connection. If the local node is supporting **NOTIFY**, the host can be configured to send a **BIND** when it receives the **NOTIFY** message sent by the local node when the application opens its SSCP connection. (For more information, see SSCP Connection.)

- A host PLU sends a **BIND** command to the SNA server LU. If the SSCP connection for the LU is not open, the local node returns a negative response to the **BIND** request. The sense code used is 0x0845 (**NOTIFY** will be sent). The local node does not open the PLU connection. In this case, the local node sends **NOTIFY** when the SSCP connection is opened. (For more information, see SSCP Connection.)

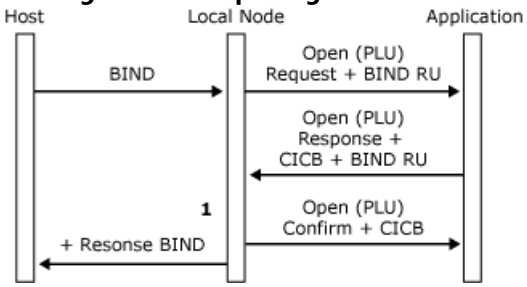
To successfully open the PLU connection, the local node sends an Open(PLU) Request to the application. The application responds with an Open(PLU) OK Response. Finally the local node sends an Open(PLU) OK Confirm to the application. This exchange of messages opens the PLU connection and establishes the PLU session. It should be noted that a successful PLU opening sequence is a three-way handshake, in comparison to the opening of the SSCP connection, which is a two-way handshake.

The **Open(PLU) Request** is delivered to the application using the SSCP connection for the LU. The **Open(PLU) Request** contains the application name and open resource identifier to allow applications to correlate the PLU and SSCP connections.

The **Open(PLU) Request** indicates the logical unit that the **BIND** request was directed to, references the resource identifier supplied in the Open(SSCP) Request for that LU, and carries the actual **BIND** request/response unit (RU) received from the host. (For more information, see Open(PLU).) It also carries the maximum RU sizes, chunk sizes (if appropriate), and pacing windows for the PLU session, to enable the application to determine the initial credit if it needs to be involved in outbound pacing. (For more information, see Pacing and Chunking.)

The message flow for a successful opening of the PLU connection (on receipt of a nonnegotiable **BIND**) is shown in the following figure. Note that the **BIND** parameters are verified (at [1]) only when the application has supplied the **BIND** check table index as part of the connection information control block (CICB).

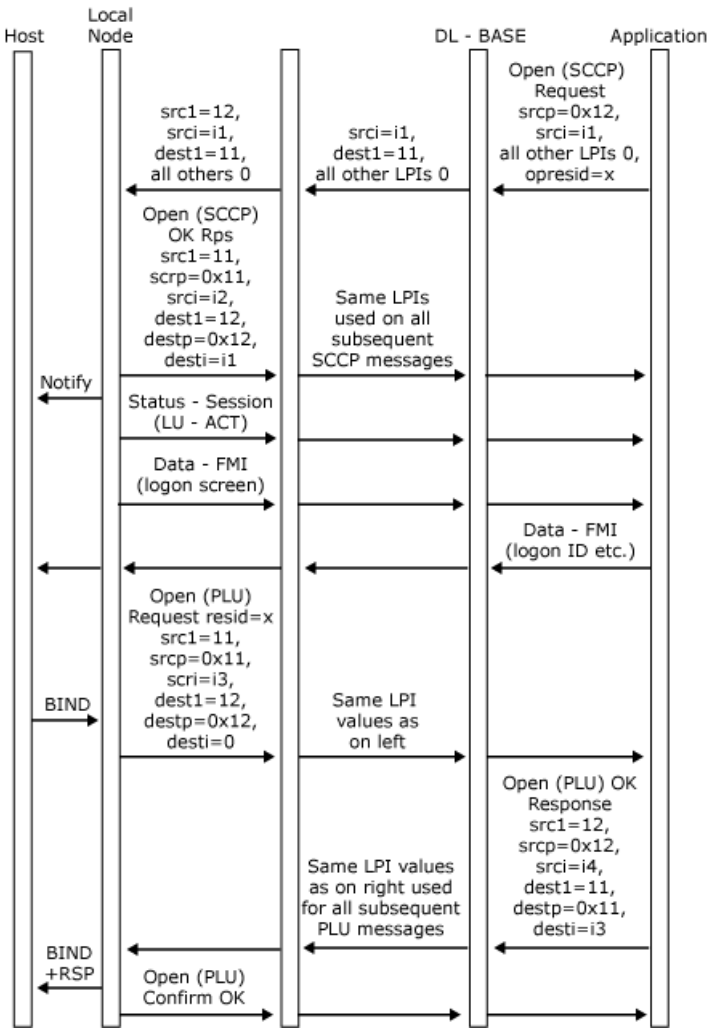
Message flow for opening a PLU connection



The following figure shows the message sequence for the initiation of both the SSCP and PLU sessions, including details of where the Locality Partner Index (LPI) values are assigned. (The application's source P value of 0x12 indicates that it is a 3270 emulator. For more information about how the source LPI values are set, see Open(SSCP) Request.) The message flow shown assumes that the connection to the host is already established and that both the configuration and the **BIND** are valid.

After this message sequence, there are two valid sets of LPI values, one for the SSCP session and one for the PLU session. The application can access either session at any time until **UNBIND** and can use the LPI values to distinguish between received data on the two sessions.

Message sequence for the initiation of both the SSCP and PLU sessions



In This Section

- BIND Checking

BIND Checking

The Open(PLU) OK Response contains the connection information control block (CICB), which enables the application to customize certain characteristics of the connection and contains information used in **BIND** verification. Note that the local node verifies the **BIND** parameters carried on the **Open(PLU) OK Response**. It does not maintain a copy of the original **BIND** request/response unit (RU) from the host. If the **BIND** is negotiable, the application is permitted to modify the parameters in the **BIND** RU, but if it is nonnegotiable the application should return the **BIND** RU unmodified. A negotiable **BIND** flag is provided in the Open(PLU) Request.

Although many characteristics of the PLU session are determined by the **BIND** parameters, the application can select certain characteristics by specifying fields in the CICB. For more information, see the following table. More detailed information about CICB usage and the effect on the PLU session of selecting various CICB options is given in context in the topics of this section that deal with PLU session characteristics such as chaining and pacing.

The **BIND** is verified using a **BIND** check table entry (whose index is specified in the CICB). The entries in this correspond to the various fields in the **BIND**. The **BIND** check table entries are stored in the configuration file. For example, the **BIND** check table entry can specify that the **BIND** be accepted if the secondary chain response protocol is either "definite response" or "definite or exception response" (byte 5 bits 2 and 3 = B10 or B11). This would be appropriate if the application did not want to send Request Exception (RQE) chains.

Connection information control block usage is shown in the following table.

Field	Explanation
Segment delivery option	A value of 0x00 indicates that the local node should perform outbound segment assembly and only deliver complete RUs. A value of 0x01 indicates that the application wants the local node to deliver RU segments. For more information, see Segment Delivery.
Application pacing option	A value of 0x00 indicates that the application requires the local node to handle pacing. A value of 0x01 indicates that the application needs to be involved with outbound pacing through Status-Resource messages. For more information, see Pacing and Chunking.
Application cancel option	A value of 0x00 indicates that the local node should automatically generate CANCEL. A value of 0x01 indicates that the application will generate CANCEL. For more information, see Inbound Chaining.
Application transaction numbers option	A value of 0x00 indicates that the application does not support transaction numbers. A value of 0x01 indicates that the application does support transaction numbers. For more information, see Recovery.
BIND check index	Gives the index of the **BIND** check table entry against which the BIND parameters should be verified. One of the following values should be used: • 0x01 —3270 printer session • 0x02 —3270 display session • 0x10 —LUA (LU type 0) application

The **Open(PLU) Confirm** from the local node to the application indicates whether the **BIND** verification was successful, and if so, supplies the bind information control block (BICB). The BICB summarizes the session **BIND** parameters in a format suitable for high-level languages and effectively defines the characteristics of the PLU session. The application not negotiating the **BIND** should usually not require to examine the **BIND** on the Open(PLU) Request and should use the BICB on the Open(PLU) OK Confirm.

The following table summarizes the fields in the BICB and their correspondence to the parameters in the **BIND** RU. For more detailed information, see the IBM manual *Systems Network Architecture: Formats,* (GA27-3136).

Position on Open(PLU) OK Confirm	Position in Bind RU [byte,bit]	Description
dataru[0]	[2,]	Function management (FM) profile

dataru[1]	[3,]	Transmission Service profile
dataru[2]	[4, 0]	Primary chaining use
dataru[3]	[4, 1]	Primary request control mode
dataru[4]	[4,2-3]	Primary chain response protocol
dataru[5]	[4, 4]	Primary two-phase commit
dataru[6]	[4, 6]	Primary compression indicator
dataru[7]	[4, 7]	Primary send End Bracket (EB) indicator
dataru[8]	[5, 0]	Secondary chaining use
dataru[9]	[5, 1]	Secondary request control mode
dataru[10]	[5,2-3]	Secondary chain response protocol
dataru[11]	[5, 4]	Secondary two-phase commit
dataru[12]	[5, 6]	Secondary compression indicator
dataru[13]	[5, 7]	Secondary send EB indicator
dataru[14]	[6, 1]	FM header usage
dataru[15]	[6, 2]	Bracket usage1
dataru[16]	[6, 2]	Bracket reset state2
dataru[17]	[6, 3]	Bracket termination rule
dataru[18]	[6, 4]	Alternate code set indicator
dataru[19]	[6, 5]	Sequence number availability
dataru[20]	[7,0-1]	Normal-flow send/receive mode
dataru[21]	[7, 7]	Half-duplex flip-flop reset
dataru[22]	[8,2-7]	Secondary pacing send window
dataru[23]	[9,2-7]	Secondary pacing receive window
dataru[24-25]*	[10,]	Secondary send maximum request unit size
dataru[26-27]*	[11,]	Primary send maximum request unit size
dataru[28]	[14,1-7]	LU-LU session type
dataru[29]	[27,]	PLU name size
dataru[30-37]	[28,]	PLU name in Extended Binary Coded Decimal Interchange Code (EBCDIC)
dataru[38]	[15,0-3]	Session type 1: PS Function Management Header (FMH) type
dataru[39]	[15,4-7]	PS data stream profile
dataru[40]	[16, 0]	Number of outstanding destinations
dataru[41]	[16, 1]	Compacted data indicator

dataru[42]	[16, 2]	Peripheral Device Information Record (PDIR) allowed indicator
dataru[43]	[15, 0]	Session type 2 or 3: query support
dataru[44]	[24,1-7]	Dynamic screen size
dataru[45]	[20,]	Basic row size
dataru[46]	[21,]	Basic column size
dataru[47]	[22,]	Alternate row size
dataru[48]	[23,]	Alternate column size

Note

10x00 = Brackets not used. 0x01 = Brackets used.

Note

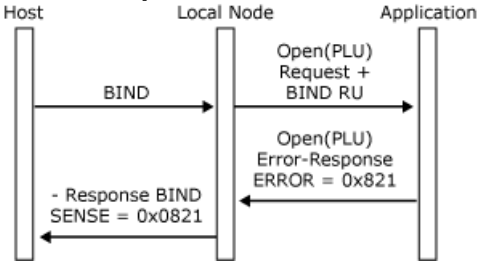
20x01 = Bracket reset state is BETB (between-brackets). 0x02 = Bracket reset state is INB (in-bracket).

Note

These values are of type INTEGER (all others are of type CHAR).

The opening PLU sequence can fail if the application rejects the Open(PLU) Request (for example, if the **BIND** parameters are unacceptable on a nonnegotiable **BIND**) by sending Open(PLU) Error Response and appropriate sense codes. The local node sends to the host a negative response to the **BIND** request containing the supplied sense codes. The PLU connection is considered to be closed after an **Open(PLU) Error Response**, and the local node does not generate an **Open(PLU) Confirm**. The following figure shows a failure to open the PLU connection (for a nonnegotiable **BIND**), due to the application rejecting the **Open(PLU) Request**.

Failure to open the PLU connection

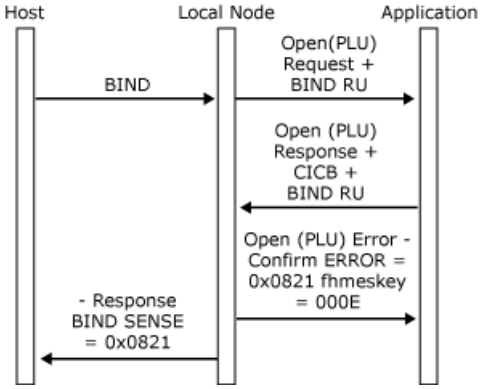


- The opening PLU sequence can also fail if the **BIND** verification against the **BIND** check table entry specified by the application fails. In this case, the local node does the following: Sends to the host a negative response to the **BIND** request with appropriate sense codes.

- Sends to the application an Open(PLU) Error Confirm with the first word of the sense codes as the first error code and the index of the **BIND** parameter in error as the second error code.

The PLU connection is considered to be closed after the **Open(PLU) Error Confirm**. The following figure shows failure to open the PLU connection due to **BIND** verification failure. Note that error code 2 gives the index in the RU of the **BIND** parameter in error.

Failure to open the PLU connection due to BIND verification failure


Host
Local Node
Application
BIND
Open(PLU)
Request +
BIND RU
Open (PLU)
Response +
CICB +
BIND RU
Open (PLU) Error -
Confirm ERROR =
0x0821 fhmeskey
= 000E
- Response
BIND SENSE
= 0x0821

Closing the PLU Connection

Either the application or the local node can terminate the primary logical unit (PLU) connection. The criteria for closing are:

- The local node closes the PLU connection if it receives an **UNBIND** request from the host PLU, which terminates the PLU session. If the **UNBIND** type is **BIND forthcoming** (0x02), the local node sets the **BIND**-forthcoming indicator in the Close(PLU) Request, so that the application can reserve any necessary resources.
- The local node closes the PLU connection if it receives a Deactivate Logical Unit (DACTLU) or Deactivate Physical Unit (DACTPU) request from the system services control point (SSCP).
- The local node closes the PLU connection if it receives an outage notification from data link control.
- The local node closes the PLU connection if it detects a critical error in a message from the application, putting the application in a critically failed state. In this case, the local node sends a **TERM-SELF** request to the host to elicit an **UNBIND**.
- The application should close the PLU connection for logical power-off conditions. For example, if its resources are temporarily unavailable, or when the user finishes using the session.

When the local node issues a Close(PLU) Request, the application can determine the reason from the **Close** control field. There may be an associated status message on either the PLU connection (a Status-Acknowledge(Nack-2)) or the SSCP connection (a Status-Session message if the LU has been deactivated).

Whether the local node or the application closes the connection, the message is the same. The initiator of the **Close** sequence sends a **Close(PLU) Request** to its partner, which responds with a **Close(PLU) Response**. The **Close(PLU) Request** is unconditional. The **Close(PLU) Response** always reports that the connection was successfully closed.

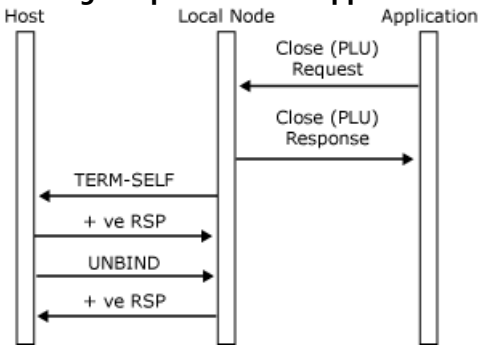
The **Close(PLU) Response** is provided so that the initiator of the **Close** sequence can determine when outstanding data and status messages have been delivered. To avoid possible race conditions, the application should discard all messages it receives on the PLU connection after issuing a **Close(PLU) Request**, including any **Close(PLU) Request** messages from the local node, until it receives the **Close(PLU) Response**.

Note that, if the application sends a Close(SSCP) Request while the PLU session is active, the local node will close the PLU connection (as if **Close(PLU) Request** had been sent) as well as the SSCP connection.

The message sequence for an application-initiated **Close** is shown in the following figure. The local node sends a **TERM-SELF** request to the host to elicit an **UNBIND**.

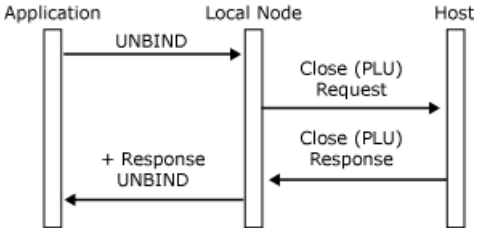
If the host generates an **UNBIND** automatically on receipt of a **TERM-SELF**, the application can view **Close(PLU)** as equivalent to the termination of the PLU-SLU session.

Message sequence for an application-initiated Close



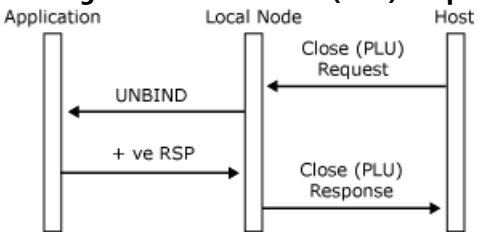
The message flow for a local node-initiated **Close** after receiving an **UNBIND** request from the host is shown in the following figure.

Message flow for a local node-initiated Close after receiving an UNBIND request



When an application is using the logical unit application (LUA) variant of the FMI, issuing a Close(PLU) Request causes the node to immediately unbind the PLU session by sending an **UNBIND** request to the PLU. The **Close(PLU) Response** is returned to the application on receipt of the **UNBIND** response, as shown in the following figure.

Message flow for the Close(PLU) Response



See Also

Reference

LUSTATs

Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Using the PLU Session

When the primary logical unit (PLU) connection is open, the application has access to the PLU session and can communicate with the host PLU.

In This Section

- PLU Session Characteristics
- PLU Session Status

PLU Session Characteristics

The local node provides support on the primary logical unit (PLU) session for function management (FM) profiles 2, 3, 4, and 7 and Transmission Service profiles (TS profiles) 2, 3, 4, and 7. Support of these profiles means that the local node supports LU-LU session types 0, 1, 2, and 3. Using the PLU connection, the application can send and receive any FM data that is valid for these LU-LU types.

The protocols appropriate to a particular session are determined by the parameters in the **BIND** request that establishes the session. The **BIND** parameters are reported to the application in the bind information control block (BICB) on the Open(PLU) OK Confirm message. It is the application's responsibility to conform to the session protocols reported in the BICB.

Due to the wide range of **BIND** parameters allowable on a session and the options available to an application in the CICB on the Open(PLU) OK Response, this section does not attempt a complete description of the protocols for a particular session. The remaining topics in this section describe the general protocol characteristics of the PLU session, such as chaining, brackets, and so on.

Most of the protocol descriptions in the remainder of this section are accompanied by figures to illustrate the important features. The figures show:

- The relevant response header flags in SNA requests/responses.
- The sequence number of SNA requests/responses.
- Any sense data (shown as "SENSE=...") on SNA responses or **Data** messages.
- The acknowledgment required (ACKRQD) field in Data and Status-Control messages.
- The relevant application flags in **Data** and **Status-Control** messages. (For more information, see Application Flags.)
- The message key field in **Data** messages.
- Any error codes (shown as "ERROR=...") in Status-Acknowledge or **Status-Control** messages.

For simplicity, it is assumed that all messages are function management data flowing on the same PLU session that:

- Uses half-duplex flip-flop protocols.
- Uses brackets, with reset state of between-bracket.
- Does not use the PLU CICB segment delivery option. (For more information, see Segment Delivery).

See Also

Reference

LUSTATs

Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

PLU Session Status

While the primary logical unit (PLU) connection is open, the local node reports any changes of state to the application through Status-Session messages. There is only one **Status-Session** status code that can occur on the PLU connection, which is listed in the following table.

Status code	Description
BETB	The PLU session has made the transition from the in-bracket state to the between-bracket state. (For more information, see Brackets.)

Status-Session Codes describes the **Status-Session** status codes.

See Also

Reference

LUSTATs

Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Outbound Chaining

The local node checks that outbound chains of requests conform to the correct SNA usage, to the chaining usage for the session, and to the current state of the session. The local node will accept valid outbound chains of data from the host if one of the following is true:

- Data traffic is active on a full-duplex session.
- The session is in a state where it can receive data.
- The session is between brackets with neither half-session currently sending, or the session is in contention for a half-duplex contention session. (For more information, see Brackets.)
- The session is waiting for the host to initiate a recovery procedure. For example, the local node has sent a negative response to an outbound chain. (For more information, see Recovery.)

The local node sends a Data message to the application for each outbound request, but note the effects of the application specifying the segment delivery option in the connection information control block. (For more information, see Segment Delivery.) If the application does not specify segment delivery, the begin chain indicator (BCI) and end chain indicator (ECI) application flags in the message header reflect the chaining indicators in the request header of the request.

An outbound chain can terminate in several ways:

- The chain is received complete and without error. All the requests in the chain have been passed to the application as **Data** messages and have been acknowledged where applicable.
- The application detects an error in a Data message while receiving the chain. The application should send a Status-Acknowledge(Nack-1) with associated sense data to the local node, which sends a negative response plus the sense data to the host for the request corresponding to the **Data** message in error. The local node will not purge the remainder of the chain, so the application will see End Chain (EC). Alternatively, the host can terminate the chain with a CANCEL, which is delivered to the application as a **Status-Control(CANCEL)** with ACKRQD set.
- The local node detects an error in a request and presents the application with a system detected error **Data** message to report the premature termination of the chain. This message carries the system detected error indicator (SDI) and ECI application flags, the sense codes for the error, and the ACKRQD indicator. It does not carry user data. When the application responds with Status-Acknowledge(Ack), the local node generates a negative response to the chain using the appropriate sense code. The application can use the reported sense codes to generate diagnostic information for the user. (For example, a 3270 emulator would generate **PROG** check codes.) The local node will purge the remainder of the chain, so the application may not see EC. Alternatively, the host can terminate the chain with a **CANCEL**, which is delivered to the application as a **Status-Control(CANCEL)** with ACKRQD set.
- The host can cancel the chain while sending, by sending the **CANCEL** request. The local node sends a **Status-Control(CANCEL)** message to the application, which the application must acknowledge.

If an error occurs while receiving a chain, and the session uses half-duplex flip-flop protocols, the application must assume an error-recovery-pending state. (For more information, see Recovery.)

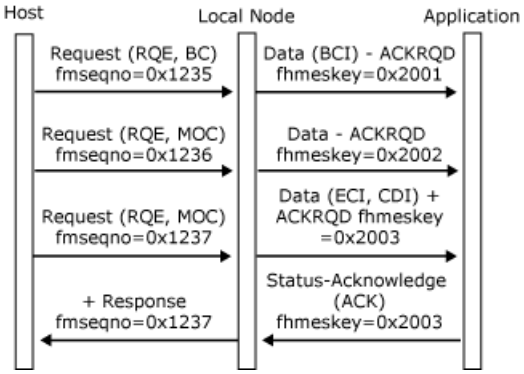
For a session using half-duplex flip-flop protocols, if the application flags in the last Data message of the chain have the CDI (change direction) flag set:

- If the chain was received without error, the application has direction.
- If the application rejected any message in the chain, the host retains direction.

The following four figures illustrate outbound chaining protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

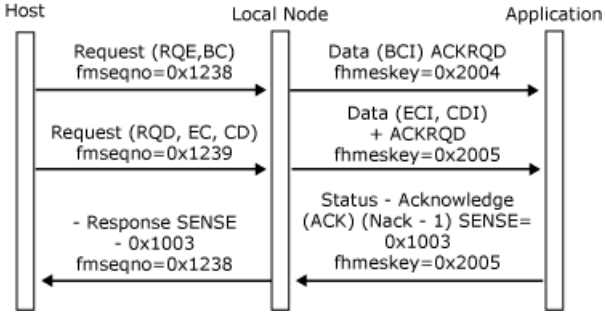
In the first figure, a complete outbound chain is received without error and accepted by the application. Note that after sending **Status-Acknowledge(Ack)**, the application has direction.

Outbound chain received without error and accepted by the application



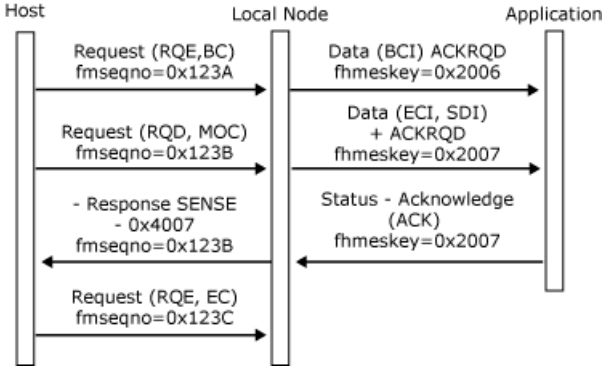
In the following figure, a complete outbound chain is received without error, but is rejected by the application. Note that even though the chain carried CD, the application does not have direction.

Outbound chain received without error, but is rejected by the application



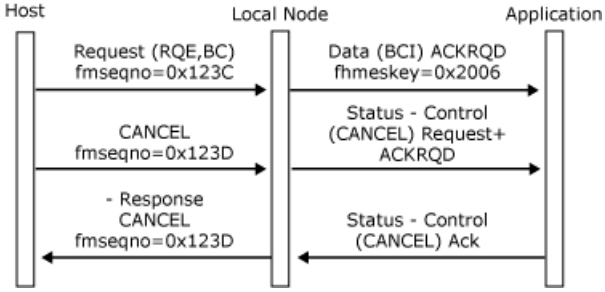
In the following figure, the local node detects the invalid use of RQD without EC and converts the request to a **Data** message with the SDI application flag set, plus ACKRQD and appropriate sense codes. The application's **Status-Acknowledge(Ack)** drives the negative response to the host. This example assumes that the receive check 4007 has been specified in the CICB on the **Open (SSCP)**.

Local node detects invalid use and converts request



In the following figure, the host cancels the outbound chain.

Host canceling the outbound chain



See Also

Reference

LUSTATs

Concepts

Inbound Chaining

Segment Delivery

Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Inbound Chaining

The division of application data into Data messages and the control of inbound chaining are the responsibility of the application.

The secondary maximum send request unit size for the session is a parameter in the **BIND** from the host and is available in the bind information control block (BICB) on the Open(PLU) OK Confirm message. The application should ensure that each inbound **Data** message corresponds to a single request unit. It does not contain more data than the maximum request unit size given in the BICB.

The application should use the begin chain indicator (BCI) and end chain indicator (ECI) application flags in the **Data** message headers to control chaining. (For more information, see Application Flags.) The chain is the unit of recovery, and if recoverable errors occur in the chain, the application should assume responsibility for recovery. (For more information, see Recovery.)

An inbound chain can terminate in the following ways:

- The complete chain is sent without errors. All the **Data** messages in the chain have been passed to the host. If the session allows the secondary to send definite-response chains, and the application sets the **ACKRQD** field in the last **Data** message of the chain, the application receives a Status-Acknowledge(Ack) from the local node when the host supplies a response.

- The local node detects a critical error in the format of a **Data** message from the application or in the state of the session. The local node rejects the **Data** message with a Status-Acknowledge(Nack-2) containing an error code and closes the PLU connection. Note that the local node will generate an inbound **CANCEL** request before closing the PLU connection. The local node will send a **TERM-SELF** request to the host to elicit an **UNBIND**.

- The host sends a negative response to a request in the chain. The local node sends a Status-Acknowledge(Nack-1) message to the application with the sense codes and sequence number from the negative response. If the host rejects a request that does not carry the ECI application flag, and the application does not specify the application cancel option in the PLU CICB, the local node also generates an inbound CANCEL request. When the application specifies application cancel, it must send EC or **Status-Control(CANCEL)** to terminate the chain. Any subsequent inbound chains are rejected with a noncritical **Status-Acknowledge(Nack-2)**, sense code 0x2002 or 0x2004 (chaining or direction). When the application receives the **Status-Acknowledge(Nack-1)** message, it should stop sending data after this chain for half-duplex flip-flop sessions because the direction has passed to the host. (For more information, see Direction.)

- The application cancels the chain while sending, by sending a **Status-Control(CANCEL)** message to the local node. The local node sends a CANCEL request to the host and sends a **Status-Control(CANCEL)** Acknowledge to the application on receiving a positive response from the host. Responses from the host to requests sent before the **CANCEL** will generate appropriate **Status-Acknowledge** messages to the application if the original Data messages had the **ACKRQD** field set.

- The application closes the PLU connection while sending the chain. The local node sends a **Close(PLU) Response** to the application. Responses from the host to requests sent before the **Close(PLU)** message will not generate **Status-Acknowledge** messages to the application. Note that the local node will also generate an inbound **CANCEL** request and a **TERM-SELF** request to elicit an **UNBIND**.

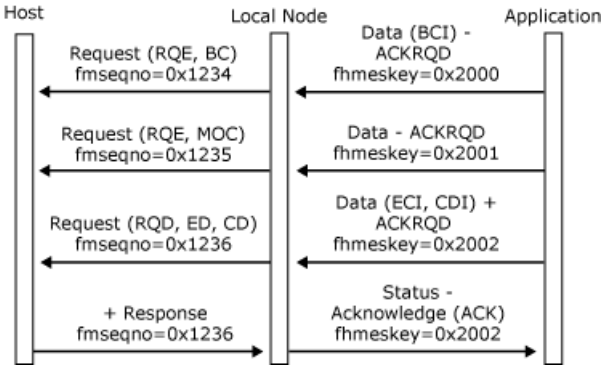
If the local node detects a noncritical error in the format of a **Data** message from the application or the state of the session, it does not close the PLU connection. Instead, it rejects the **Data** message in error with a **Status-Acknowledge(Nack-2)** containing an appropriate error code. No data is sent to the host.

If an inbound chain terminates with an error, when the session uses half-duplex protocols, the application must assume a receive state. (For more information, see Recovery.)

The following six figures illustrate inbound chaining protocols between the local node and the application, and how those protocols relate to the underlying SNA protocols.

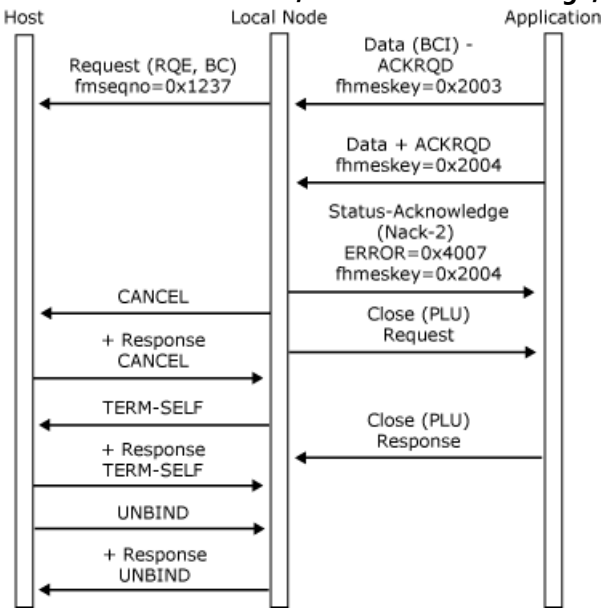
In the first figure, a complete inbound chain is sent without error and accepted by the host. Note that after receiving **Status-Acknowledge(Ack)** the application relinquishes direction to the host.

Inbound chain is sent without error and accepted by the host



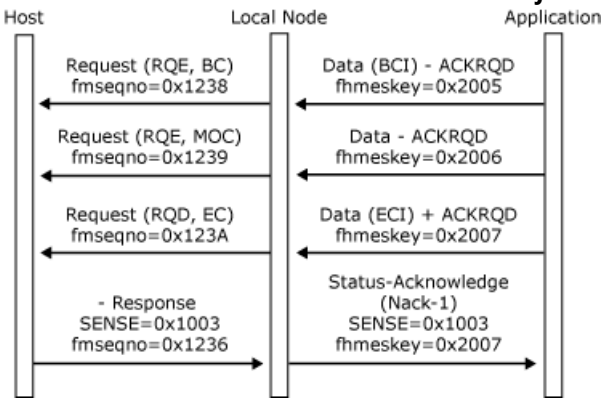
In the following figure, the local node detects a critical error in the format of the second **Data** message in the chain (**ACKRQD** without the ECI application flag), sends a **Status-Acknowledge(Nack-2)** to the application with the appropriate error code, and closes the PLU connection. Note that the local node only generates the **CANCEL** if the session's function management (FM) profile supports **CANCEL**.

Local node detects error, sends a Status message, and closes the PLU connection



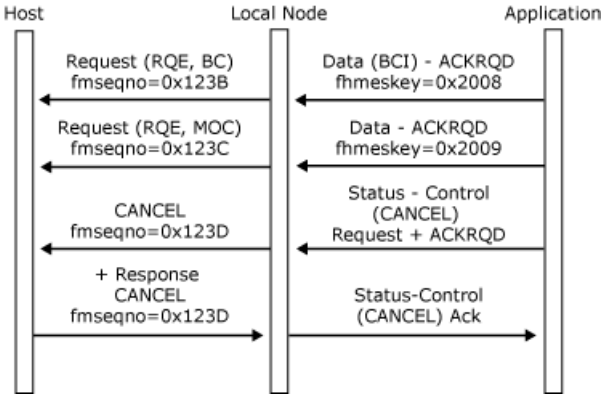
In the following figure, a complete inbound chain is sent without error, but is rejected by the host. After the negative response, the application must enter receive state, pending error recovery. (For more information, see Recovery.)

Inbound chain is sent without error but is rejected by host



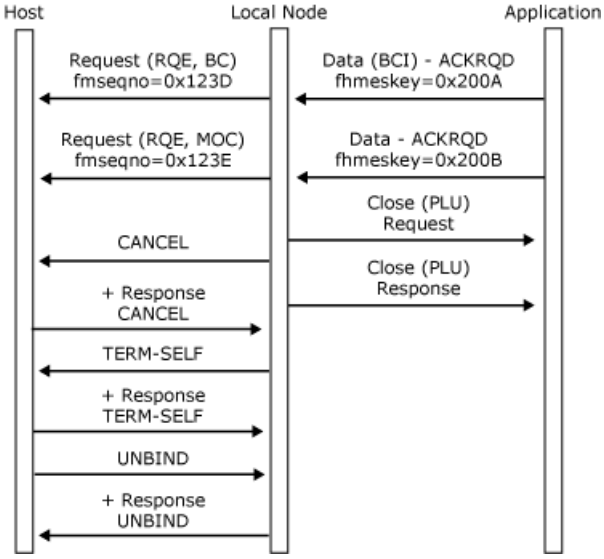
In the following figure, the application cancels the chain by sending **Status-Control(CANCEL)**. Note that the application still has direction and can start a new chain.

Application cancels the chain with a Status-Control(CANCEL)



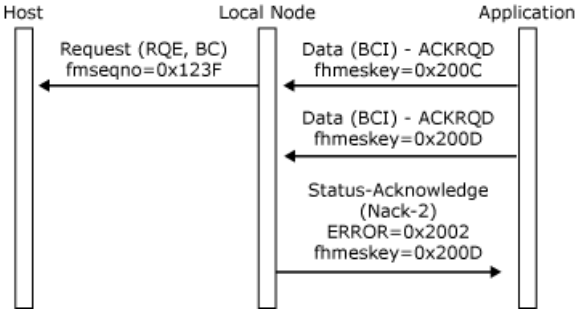
In the following figure, the application closes the PLU session while sending the chain. The local node only generates the **CANCEL** if the session's FM profile supports **CANCEL**.

Application closes the PLU connection while sending the chain



In the following figure, the local node detects a noncritical error in the format of the second **Data** message in the chain and sends a **Status-Acknowledge(Nack-2)** to the application with the appropriate error code.

Local node detects a noncritical error and sends a Status-Acknowledge(Nack-2)



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Segment Delivery

If the maximum request/response unit (RU) size for a session (supplied in the **BIND** parameters) allows RUs that are larger than the maximum size of a data link control transmission unit, for example, a Synchronous Data Link Control (SDLC) frame, the local node's path control is responsible for assembling outbound segments into RUs and segmenting inbound RUs where required.

However, certain IBM products (for example, SNA models of the 3270 controllers) do not perform outbound segment assembly, to improve perceived response times at display terminals by displaying each segment as soon as it is received. This feature is referred to as window shading.

The local node allows an application to specify a segment delivery option in the connection information control block (CICB) on the Open(PLU) OK Response. If an application specifies this option, the local node's path control does not assemble outbound segments into complete RUs, and the local node delivers the segments to the application in Data messages. This enables an application emulating a 3270 device to reproduce the perceived response characteristics of the IBM device. In cases where throughput is high, such as 3270 file transfer, segment delivery can give improved performance compared to RU delivery.

Note that there is no comparable feature for inbound data. The application must present **Data** messages containing complete RUs to the local node. Also, there is no support for segment delivery on the system services control point (SSCP) session and connection (where the maximum RU size is limited to 256 bytes).

The local node supports the segment delivery option in such a way that the constraints placed on an application receiving data in either form are identical. If complete RUs are required, the local node rebuilds the RUs from segments in path control. If segments are required, the local node handles all segmentation indicators and modifies processing within its SNA layers to cater for segmented RUs.

All **Data** messages delivered to the application contain application flags, whereas only the first segment in an RU contains a response header (RH). The local node delays the end chain (EC) and change direction (CD) indicators if they occur in the RH of the RU's first segment, and sets the corresponding ECI and CDI application flags in the **Data** message corresponding to the last segment of the RU. Therefore, the **Data** messages corresponding to RU segments have application flags set as if they corresponded to whole RUs. This considerably simplifies the handling of chaining, bracket, and half-duplex protocols for an application using the segment delivery option.

Note
EB is not delayed until end basic information unit (EBIU), because the application should use the Status-Session between-brackets message to determine when to enter the between-brackets state.

See Also
Reference
LUSTATs
Concepts
Outbound Chaining
Inbound Chaining
Application-Initiated Termination
Response Time Monitor Data
Other Resources
Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Brackets

This section primarily describes the bracket protocols between the local node and an application for a session supporting half-duplex flip-flop with brackets.

The local node enforces no bracket protocols for full-duplex sessions. Therefore, messages with begin bracket (BB) are not presented as **Status-Control(BID)** messages, and there are no **Status-Session(BETB)** messages.

The management of this protocol for a generalized application is complex, and there is a significant amount of code in the local node to simplify the application's perception of the protocol. An application is only aware of two states:

- In-bracket
- Between-bracket

The local node, in addition to the states of in-bracket and between-bracket, maintains transient states with a large state transition matrix, or finite-state machine, governing the half-session's state at a particular time.

In This Section

Bracket Initiation

While a session is in the between-bracket state, contention exists. Either the application or the host primary logical unit (PLU) can attempt to initiate a bracket, as follows:

- The application initiates a bracket by sending a Data message with the begin bracket indicator (BBI) application flag and **ACKRQD** set while in the between-bracket state. The local node sends a request corresponding to the **Data** message to the host PLU. The application has successfully initiated a bracket and is in the in-bracket state. Flip-flop protocols are now in force until the bracket is terminated.

- The application initiates a bracket by sending a **Status-Control(LUSTAT)** with the BBI application flag set while in the between-bracket state. The local node sends an **LUSTAT** request to the host PLU. The application has successfully initiated a bracket and is in the in-bracket state. Flip-flop protocols are now in force until the bracket is terminated.

- The host PLU sends a **BID** request while in the between-bracket state. The local node sends a **Status-Control(BID)** with **ACKRQD** to the application. (For more information, see Status-Control Message.) The application replies with a **Status-Control(BID) Acknowledge**, to indicate that it is willing to accept a bracket. The local node sends a positive response to the **BID** request. The host PLU has successfully initiated a bracket, and the application's state is in-bracket, with flip-flop protocols applying until the bracket is terminated.

- The host PLU sends data in an RU carrying the BB indicator in the RH while in the between-bracket state. The local node presents this method of initiating a bracket in the same way as if the host PLU had initiated the bracket with **BID**. The local node sends a **Status-Control(BID)** with **ACKRQD** to the application. The application replies with a **Status-Control(BID) Acknowledge** to indicate that it is willing to accept the bracket. The local node sends the Data message corresponding to the RU to the application and sends a positive response to the data RU. The host PLU has successfully initiated a bracket, and the application's state is in-bracket, with flip-flop protocols applying until the bracket is terminated.

- The host PLU sends an **LUSTAT** request carrying the BB indicator in the RH. The local node presents this method of initiating a bracket in the same way as if the host PLU had initiated the bracket with **BID**. The local node sends a **Status-Control(BID)** with **ACKRQD** to the application. The application replies with a **Status-Control(BID) Acknowledge** to indicate that it is willing to accept the bracket. The local node sends a **Status-Control(LUSTAT)** to the application, which requires an acknowledgment. The host PLU has successfully initiated a bracket, and the application's state is in-bracket, with flip-flop protocols applying until the bracket is terminated.

- The host attempts to initiate a bracket using a **BID** request or an RU carrying BB, which the local node sends to the application as a **Status-Control(BID)**, but the application cannot accept the bracket. The application should send a negative **Status-Control(BID)** response with an appropriate sense code. The local node sends a negative response to the BID carrying the sense code supplied by the application. The application's state is still between-bracket. The application should use one of the following sense codes:

    - 0x081B if it has already committed resources for an inbound transfer. For example, a terminal operator has begun typing.

    - 0x0814 if it currently cannot begin a bracket but will notify the host when resources become available. For example, a 3270 printer is being used for local copy in between-bracket printer sharing mode. At a later stage when the resources become available, the application should temporarily reserve the resources and send a **Status-Control(RTR)** to the local node. If the host rejects the **RTR**, the local node returns a **Status-Control(RTR) Negative-Acknowledge-1** response, and the application can release the resources. Otherwise, the host attempts to initiate a bracket that the application must now accept.

- If the application has successfully initiated a bracket, a bracket race may occur due to the host PLU attempting to initiate a bracket. The application gets a **Status-Control(BID) Request**, which it should reject with 0x080B or 0x0813. The application retains direction after race negative responses. (For more information, see Recovery.) The application's bracket state remains as in-bracket.

The application needs to be aware of one further complication in bracket initiation. All the cases relate to sessions whose bracket reset state is between-bracket. A state of contention exists, and either half-session can attempt to begin a bracket.

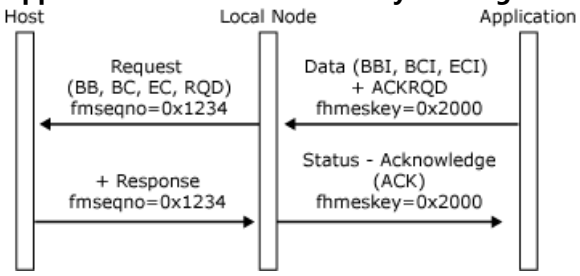
However, the **BIND** parameters for the session can specify a bracket reset state of in-bracket. If the bracket reset state is in-bracket, one half-session is considered to have already successfully initiated a bracket. Flip-flop protocols will then apply until a **Status-Session(BETB)** is received, at which time the session reverts to a contention state and bracket initiation proceeds as described earlier.

The application must set its bracket state when the PLU connection is opened (on receipt of the Open(PLU) OK Confirm message) and reset it each time the session is reset (after receipt of a **Status-Control(CLEAR) Request**). The appropriate bracket reset state for the session is supplied to the application in the BICB on the Open(PLU) OK Confirm message.

The following six figures illustrate bracket initiation protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

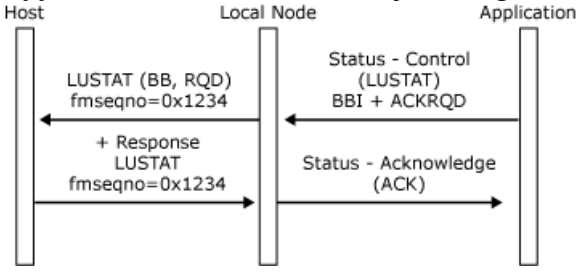
In the first figure, the application initiates a bracket by sending an inbound chain with the BBI application flag set when its state is between-bracket. The application's state is in-bracket until it receives a **Status-Session(BETB)**. (If the application can send RQE chains, a bracket can be opened by sending an RQE chain.)

Application initiates a bracket by sending an inbound chain



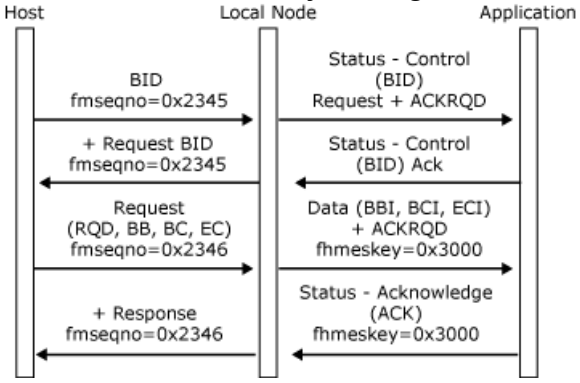
In the following figure, the application initiates a bracket by sending a **Status-Control(LUSTAT)** with the begin bracket indicator (BBI) application flag set when its state is between-bracket. The application's state is in-bracket until it receives a **Status-Session(BETB)**. The **LUSTAT** can be sent **NOACKRQD** (RQE) if required.

Application initiates a bracket by sending a Status-Control(LUSTAT)



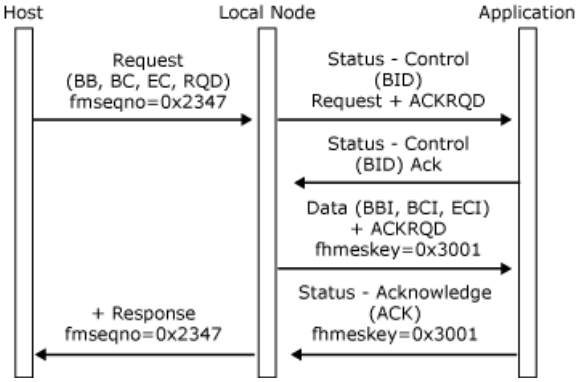
In the following figure, the host initiates a bracket by sending **BID**, which the application accepts. The application's state is in-bracket and the host can send.

Host initiates a bracket by sending BID



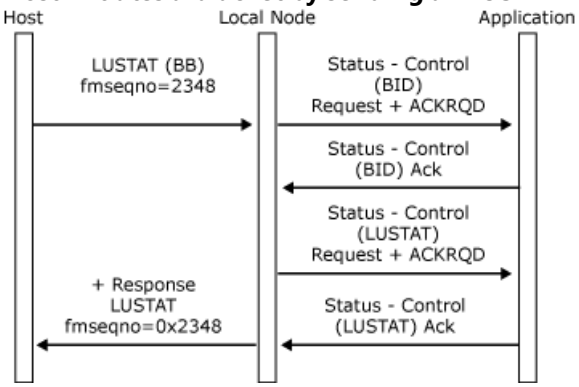
In the following figure, the host PLU initiates a bracket by sending a request with begin bracket (BB), which the application accepts. The application's state is in-bracket, and the host can send.

Host PLU initiates a bracket by sending a request with BB



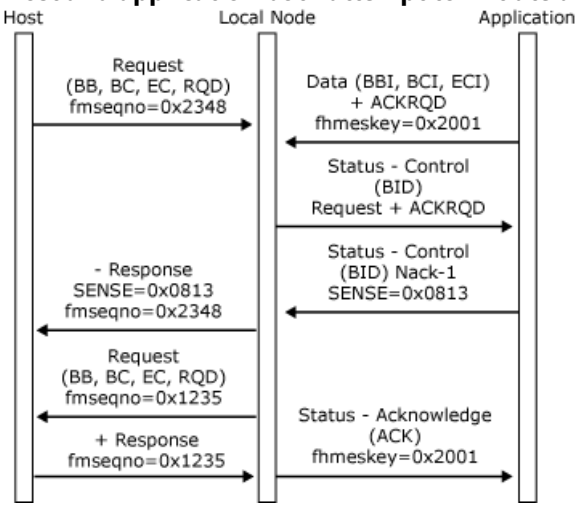
In the following figure, the host initiates a bracket by sending an **LUSTAT** with BB, which the application accepts. The application's state is in-bracket, and the host can send.

Host initiates a bracket by sending an LUSTAT with BB



In the following figure, the host and application both attempt to initiate a bracket in between-bracket state. The application rejects the host bids with sense code 0x0813, and the local node delivers the application's data. After sending the data, the application's state is in-bracket, and the application can send.

Host and application both attempt to initiate a bracket in between-bracket state



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Bracket Termination

The local node supports bracket termination rule one (conditional) and bracket termination rule two (unconditional), as specified in the **BIND** request. Some sessions only allow bracket termination by one session partner. This is a **BIND** option, supplied in the bind information control block (BICB) on Open(PLU) OK Confirm), and it is the application's responsibility to determine if (and when) it should request bracket termination.

If an application is allowed by its **BIND** to terminate brackets, it does so by setting the End Bracket Indicator (EBI) application flag in an inbound Data or **Status-Control(LUSTAT/CHASE/QC/CANCEL)** message. The bracket is only terminated when the application receives a Status-Session (BETB) from the local node.

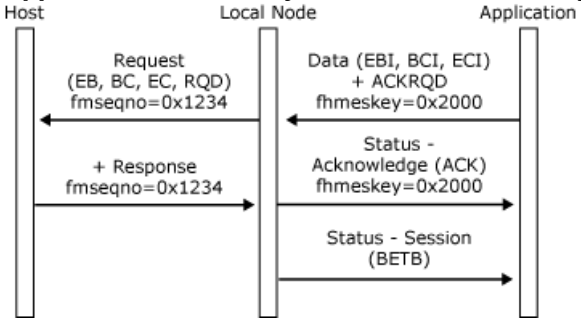
If the host terminates a bracket successfully, the local node sends a **Status-Session(BETB)** to the application. Note that the EBI application flag on outbound messages does not indicate bracket termination, but indicates that the corresponding request/response unit (RU) carried End Bracket (EB). The bracket is only terminated when the application receives **Status-Session(BETB)**.

Note that if the application queues data, it should also queue **Status-Session(BETB)** messages. They must not be processed as expedited.

The following two figures illustrate bracket termination protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

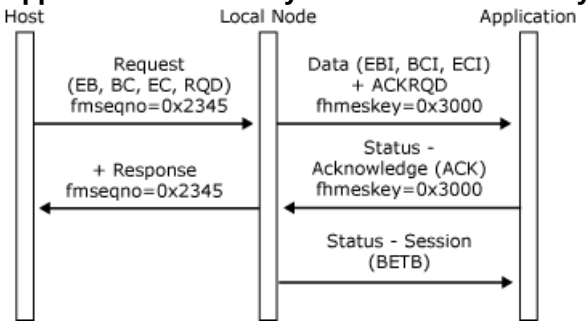
In the following figure, the application successfully terminates a bracket by sending an EBI data chain when the application's state is in-bracket, which the host accepts. The local node sends a **Status-Session(BETB)** to indicate that the application's state is now between-bracket.

Application successfully terminates a bracket by sending an EBI data chain



In the following figure, the host successfully terminates a bracket by sending an EBI data chain when the application's state is in-bracket, which the application accepts. The local node sends a **Status-Session(BETB)** to indicate that the application's state is now between-bracket.

Application successfully terminates a bracket by sending an EBI data chain



See Also

Reference

LUSTATs

Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Using the PLU Session

Direction

When an function management interface (FMI) application is communicating on its primary logical unit (PLU) connection with a normal flow request mode other than full-duplex (that is, half-duplex flip-flop or half-duplex contention), it must obey the SNA direction protocol. These two modes are treated separately.

In This Section

- Half-Duplex Flip-Flop Direction
- Half-Duplex Contention

Half-Duplex Flip-Flop Direction

The **BIND** used to establish the session carries information about the initial state of the bracket and direction machines. This can be specified in the **BIND** if either of the following conditions are satisfied:

- Brackets are not used.
- Brackets reset state is in-bracket.

If neither of the conditions hold, the initial direction state is contention.

When the direction is specified in the **BIND**, the application should assume the direction state specified in the half-duplex reset state as soon as data can flow. This field can be obtained indirectly by using a **BIND** check index that only accepts a particular direction, or directly by reading the **HDXRSET** field in the bind information control block (BICB) on the Open(PLU) OK Confirm message or by reading the **BIND** on the Open(PLU) Request. For more information about opening the PLU connection, see Opening the PLU Connection.

When in contention state, either the PLU or the application can initiate a bracket. (For more information, see Brackets.) The successful initiator of the bracket obtains direction unless direction is relinquished when opening the bracket by sending Begin Bracket (BB), Begin Chain (BC), End Chain (EC), or Change Direction (CD). Because the secondary is assumed to be the contention winner, the application can assume send state from contention sending BB and rejecting any subsequent **Status-Control(BID) Request** from the local node before receiving **Status-Session(BETB)**. When the application accepts a S**tatus-Control(BID) Request** in contention state, it must assume receive state.

Half-duplex flip-flop direction can change through the following actions:

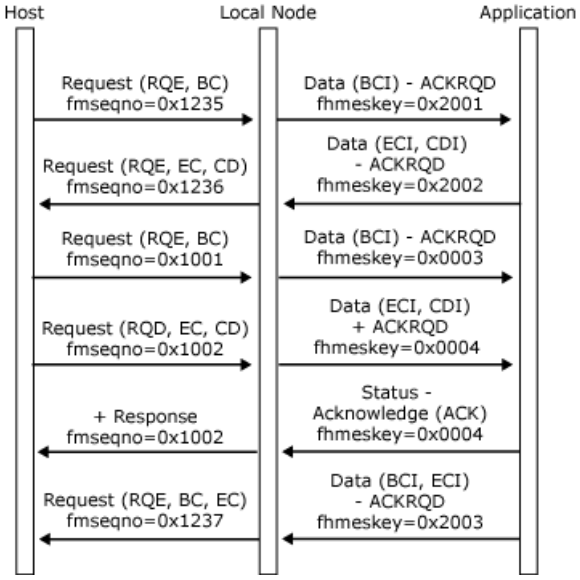
- Sending or receiving data with the change direction (CD) indicator in the RH, and the corresponding change direction indicator (CDI) flag on the **DATAFMI** and Status-Control messages. Note that CD is only used at the end of a chain (and for applications receiving segments that will be delivered with ECI, EBIUI). Also note that CD is valid on the following normal flow **Status-Control** requests: **LUSTAT**, **CANCEL**, **CHASE** and **QC**.
- Receiving a negative response when the application should assume receive state (error recovery pending state). For more information, see Recovery.
- If the application rejects data from the host carrying CDI, it must remain in receive state.

Providing the FM profile is correct (3, 4, or 7), the application can request direction from the host using a **Status Control(SIGNAL) Request** with CODE1 set to 0x0001. CODE2 is set to a user-defined value.

The following three figures illustrate the direction protocol for applications using the half-duplex flip-flop mode.

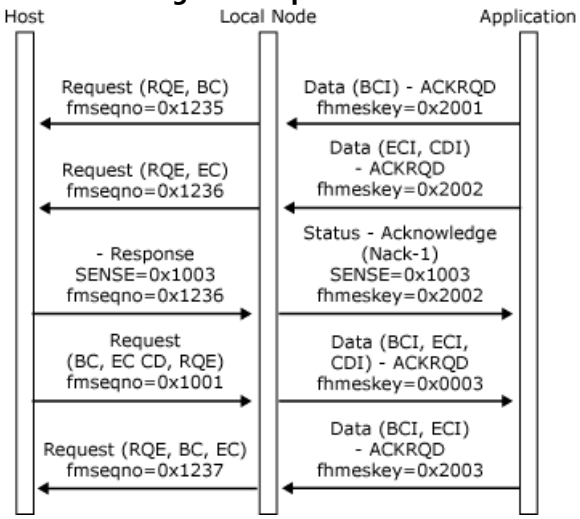
In the first figure, the application issues and receives the CD without error.

Application issues and receives the CD without error



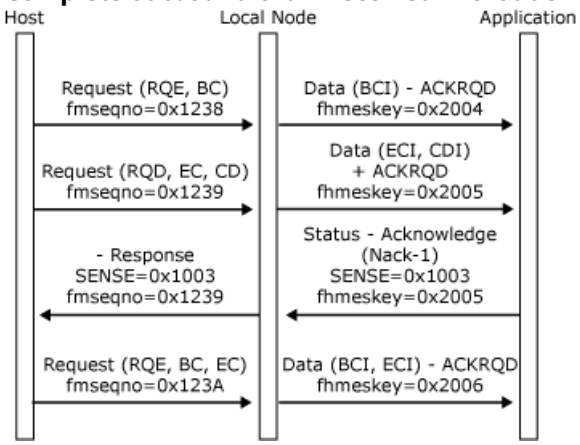
In the following figure, the host sends a negative response to inbound data. The application assumes receive state, and then the host sends CD to give the application direction.

Host sends negative response to inbound data



In the following figure, a complete outbound chain is received without error, but is rejected by the application. Note that even though the chain carried CD, the application does not have direction.

Complete outbound chain received without error, but is rejected by application



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Half-Duplex Contention

For half-duplex contention, the initial direction state is contention. Half-duplex protocol operates during a chain (only one partner can send), but the direction state normally returns to contention at the end of each chain. The change direction indicator (CDI) in the response header (RH) is thus not required. However, if the CDI is used, direction is reserved for the receiving half-session. Therefore, if the application receives change direction (CD), it should assume send state and not expect to receive data. Conversely, if the application sends CD, it cannot send again until it has received a chain from the host.

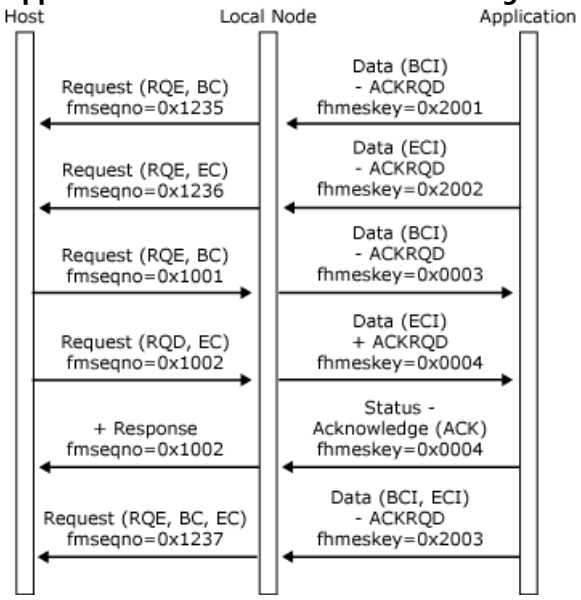
In the event of an error being discovered by either half-session, the application must assume receive state, because the host is responsible for recovery.

If both half-sessions attempt to start a chain when the direction state is contention, the race is resolved in favor of the secondary application using a sense code of 0x081B. However, the possible window between the local node and the application means that the local node cannot determine when outbound Request Exception (RQE) data is received by the application. Therefore, if the local node receives data from the application while it determines that the half-duplex contention state is receive, it will reject it with a noncritical NACK-2 (0x2004 direction).

The following two figures illustrate the direction protocol for applications using half-duplex contention mode. The three figures in the previous topic would also be valid although CD does not need to be specified.

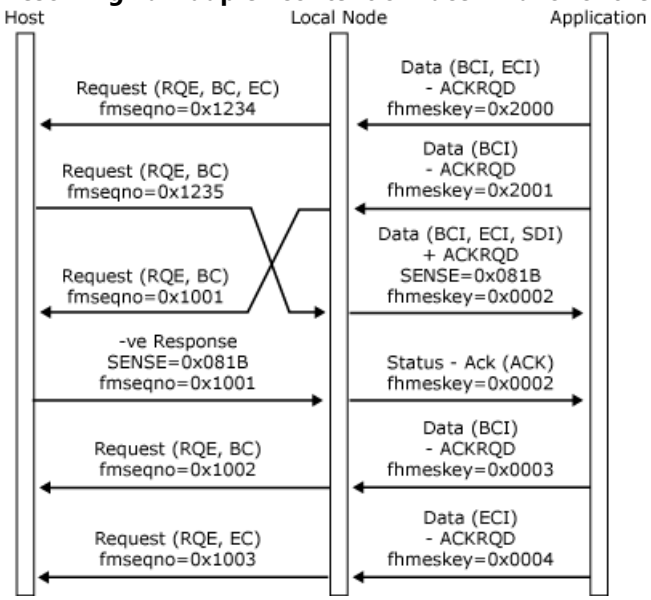
In the following figure, the application issues and receives data using half-duplex contention protocol without error.

Application issues and receives data using half-duplex contention protocol without error



In the following figure, the half-duplex contention race is resolved in favor of the application.

Resolving half-duplex contention race in favor of the application



See Also

Reference
LUSTATs
Concepts
Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data
Other Resources
Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Pacing and Chunking

The local node supports session pacing inbound and outbound, according to the pacing values in the **BIND** parameters for the session. The application can be involved in outbound pacing through the use of the Status-Resource message. Inbound pacing is handled transparently by the local node and need not concern the application.

In This Section

- Outbound Pacing
- Chunking

Outbound Pacing

If the application has enough resources to handle outbound data as fast as the network can provide it (for example, a screen), or if a higher level protocol (for example, immediate request mode) constrains the data flow, the application need not be involved in pacing, and it is possible for the local node to handle outbound pacing transparently.

However, certain types of applications may require involvement in outbound pacing. If the application has limited resources (for example, a printer), the application should specify the application pacing option in the connection information control block (CICB) on the Open(PLU) OK Response. (For more information, see Opening the PLU Connection.) The application should also provide the local node with information about the state of these resources initially on the **Open(PLU) OK Response** and periodically using **Status-Resource** messages.

To assist the application in calculating the initial credit field in the **Open(PLU) OK Response**, the local node delivers the pacing window sizes and the primary and secondary maximum request/response unit (RU) sizes on the Open(PLU) Request. The initial credit must be at least as large as the primary to secondary pacing window size. Otherwise, the **BIND** will be rejected and the application will be sent an Open(PLU) Error Confirm message. The local node fills in a suggested initial credit value of one more than the pacing window (to try to avoid stop-start situations).

Note that the local node will also reject the **BIND** if the application specifies that it needs to be involved in pacing (of whatever initial credit), but the **BIND** specifies that there is no outbound pacing.

Only function management data (FMD) requests are part of the credit scheme, so the application must maintain space within its buffer for one Status-Control request per RU in addition to the number of RUs specified by the initial credit count. (A **Status-Control** message takes up 36 bytes.)

Each unit of credit that the application delivers to the local node allows the local node to give the application a single RU (or a single chunk if chunking is being used). Note that if the application is receiving segments, this may correspond to several **DATAFMI** messages. The application can count RUs for the purpose of outbound flow control by using the begin basic information unit (BBIU) and end basic information unit (EBIU) flags.

The application should maintain a credit-used count, which it should report to the local node on Status-Resource messages. The application needs to take the following actions:

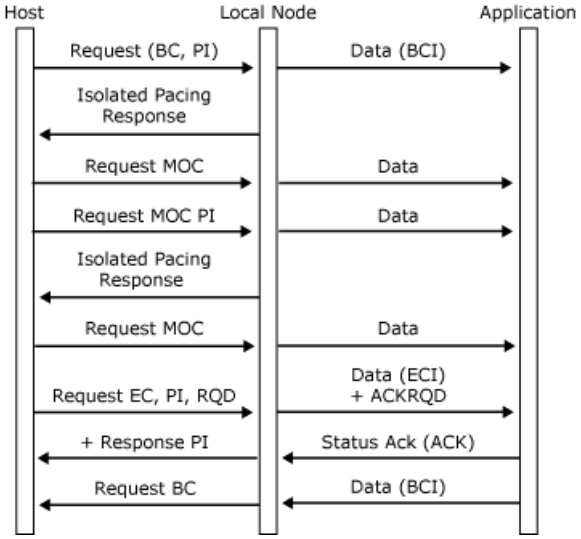
- On processing (not receiving) **DATAFMI** messages with EBIU set (corresponding to FMD requests), increment the credit-used count by one.
- On processing **Status-Control** messages and all other messages from the local node, do not increment the credit-used count.
- Periodically report the current credit-used count on a **Status-Resource** message.
- Report the credit-used count when its buffer becomes empty (whatever the last message processed was), unless the credit-used count is zero.
- When the credit-used count is reported to the local node, reset it to zero.

The frequency at which the application provides Status-Resource messages is not architected. However, the local node will only send the application as many Data messages as it has received credit for. When the application's credit-used count reaches the initial credit value, the local node will not send any more data. The application should attempt to send a **Status-Resource** message before this happens, because if the local node cannot send a **Data** message to the application and the host is still sending requests, the local node may not be able to send a pacing response to the host when required, with a consequent degradation of performance.

If the pacing window is small, such as one or two, the application should send a **Status-Resource** after processing each **DATAFMI** message to enable the local node to send the suitable pacing response.

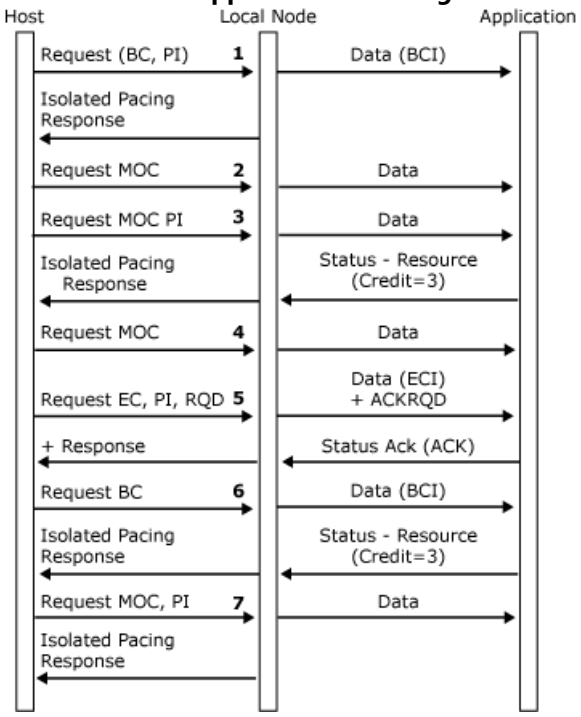
The following figure shows the local node handling outbound pacing when the application is not involved (APPLPAC = 0x00). The pacing window is assumed to be two.

Local node handling outbound pacing



The following figure shows the local node and the application handling outbound pacing with the outbound pacing window assumed to be two and the initial credit from the local node to the application assumed to be four. Note that the local node can send an isolated pacing response (IPR) to the host to get another window full of data as soon as the application has sufficient credit for the rest of the present window and the next window.

Local node and application handling outbound pacing



See Also

Reference

LUSTATs

Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Chunking

Chunking can be thought of as similar to segmentation. (For more information, see Segment Delivery.) The distinction is that segmentation is determined by the communications link between the local node and the remote system, whereas chunking is determined by the communications link between the application and the local node.

The application indicates on the Open(SSCP) Request whether it supports chunking, and, if so, the chunk size in bytes that it wants to use. The local node then uses the request/response unit (RU) size, the chunk size, and the segment size (if applicable) to determine whether chunking is necessary. It then specifies the chunk sizes used for inbound and outbound flow (which need not be the same) on the Open(PLU) Request. These values are specified in units of elements. (For more information, see Messages.) A value of zero for either of these sizes indicates that chunking is not necessary because the chunk size is not the limiting factor. Note that in chunking data, an RU will not be split in the middle of an element. This avoids data copying.

For example, assume that the local node is using an RU size of 8 kilobytes (KB) and segments of 2 KB, and the application's **Open(SSCP) Request** specifies segment delivery and a chunk size of 4 KB. Chunking will be used on inbound data flow (because the chunk size is smaller than the RU size), but is not necessary on outbound data flow (because data will be delivered in segments that are smaller than the chunk size).

If chunking is being used in either direction, all credit values specify the number of chunks that can be sent in that direction, not the number of RUs. Note that the segment delivery option is included on the **Open(SSCP) Request** to enable the local node to calculate the initial chunk credit values on the corresponding PLU connection. The application must also set this option on the **Open(PLU) Response**. If the **Open(SSCP) Request** and the **Open(PLU) Response** have different settings of this option, the setting from the **Open(PLU) Response** will be used. This can mean that the initial credit value used is not appropriate.

If session-level pacing is being used, the local node links this to the chunking credit. In particular, if the application withholds credit, the local node will delay sending a pacing response to the host, thereby applying back pressure to the host. This linkage is handled by the local node and need not concern the application.

Application flags on chunks of RUs are handled in the same way as those on segments. (For more information, see Application Flags and Segment Delivery.) In particular:

- FMHI, BCI, COMMIT, BBI, EBI, CODE, ENCRYP, ENPAD, QRI, and CEI are only set on the first chunk of an RU.
- ECI and CDI are only set on the last chunk of an RU.
- BBIUI is always set on the first chunk of an RU.
- EBIUI is always set on the last chunk of an RU.

Note that EBI is set on the first chunk of the last RU in a bracket and not on the last chunk as might be expected. This is the same behavior as for segment delivery. The application should use the **Status-Session(BETB)** message, not the EBI flag, to determine when a bracket has ended.

Chunks are identified using the segmentation flags BBIUI and EBIUI, and therefore the application cannot distinguish between chunks and segments if both segmentation and chunking are being used outbound. However, there is generally no need for the distinction. The application can perform window shading by displaying each unit of data as it is received, whether the unit of data is a segment or a chunk. (For more information, see Segment Delivery.)

Note
Previous versions of this document indicated this as a future feature. The support is enabled in Host Integration Server 2009. Applications can test the product version returned on a call to sepdgetinfo for version 1.2 or later before using the chunking system.

In some cases, the RU size used by the local node may be too large for the length of the path between the local node and an FMI application, for example, when using a 16 megabyte (MB) token-ring link, which can support 16 kilobyte (KB) frames. The local node allows an FMI application to specify that data transfer should be in smaller units, called chunks.

See Also
Reference
LUSTATs
Concepts

Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Response Time Monitor Data

Other Resources

Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Confirmation and Rejection of Data

The following topics describe conditions under which inbound and outbound data is confirmed or rejected.

In This Section

- Confirmation and Rejection of Inbound Data
- Confirmation and Rejection of Outbound Data

Confirmation and Rejection of Inbound Data

For every SNA chain of data sent or received for which responses are outstanding, such as Request Exception (RQE) or Definite Response Required (RQD), the local node maintains a correlation table entry. If the table entries become depleted, the local node will terminate the session using the most table entries. A Status-Error message (code 0x46) and a Close(PLU) Request are sent to the application, and a **TERM-SELF** message is sent to the host. Table entry shortages (inbound) can be avoided by sending change direction (CD) (for half-duplex) data, or data **ACKRQD,** or any **Status-Control(CHASE)**, or **Status-Control(LUSTAT)** with **ACKRQD**. Outbound shortages can be avoided by sending courtesy acknowledge messages as described in Opening the PLU Connection.

The local node sends chains of data to the host with their chain response mode specified as follows:

1. Definite

   If the application sends a **Data** message to the local node with the **ACKRQD** field set, and the **BIND** parameters specified that the secondary uses definite or definite/exception response mode.

2. Exception

   If the application sends a **Data** message to the local node without the **ACKRQD** field set, and the **BIND** parameters specified that the secondary uses exception or definite/exception response mode.

3. No-Response

   If the application sends a **Data** message to the local node without the **ACKRQD** field set, and the **BIND** parameters specified that the secondary uses no-response mode.

If the setting of **ACKRQD** on a Data message from the application does not reflect the chain response mode specified in the **BIND** parameters, the local node returns a Status-Acknowledge(Nack-2) indicating a noncritical error code. For example, if the application specifies **ACKRQD** but the **BIND** parameters do not permit the local node to send definite response chains.

In case 1, the application receives an acknowledgment to all function management data (FMD) chains it sends to the local node:

- Positive responses from the host are returned to the application as Status-Acknowledge(Ack) messages.
- Negative responses from the host are returned as Status-Acknowledge(Nack-1) messages carrying the SNA sense codes.
- Errors detected by the local node when attempting to send the message are returned as Status-Acknowledge(Nack-2) messages carrying the equivalent error code.

In case 2, the application only receives an acknowledgment of an FMD chain it sends to the local node for:

- Negative responses from the host, which are returned as **Status-Acknowledge(Nack-1)** messages carrying the SNA sense codes.
- Errors detected by the local node when attempting to send the message, which are returned as **Status-Acknowledge(Nack-2)** messages carrying the equivalent error code.

In case 3, the application only receives an acknowledgment of an FMD chain it sends to the local node when the node detects an error in the message and sends the application a **Status-Acknowledge(Nack-2)**. The only dissent that the host can make is to send a subsequent LUSTAT 0x400A (no response not supported) with the sequence number of the request in the sense qualifier field. This is presented to the application as a **Status-Control(LUSTAT)** as usual.

Whenever an application receives a **Status-Acknowledge(Ack)** or **Status-Acknowledge(Nack-1)**, it implicitly confirms receipt by the partner half-session in the host of all previously sent chains.

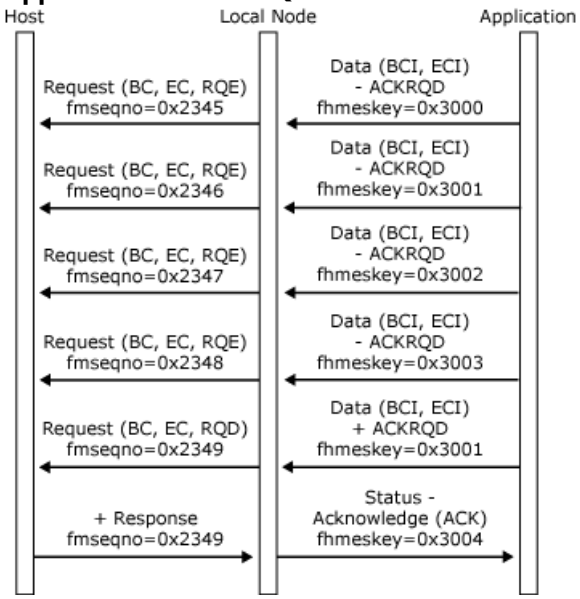
In case 2, the application does not usually receive such responses from the host to chains it has sent, and in case 3, the application never receives such responses. Therefore, to get the host to confirm receipt of all previously sent chains, the

application should issue a **Status-Control(CHASE) Request** with ACKRQD set. This causes the local node to generate an SNA **CHASE** request to the host. The receipt of the response to this CHASE confirms that the host has received this **CHASE** request and all previous chains sent by the application. The local node issues a **Status-Control(CHASE) Acknowledge** to notify the application that this is so.

The following three figures illustrate the inbound data confirmation and rejection protocols between the local node and the application, and how those protocols relate to the underlying SNA protocols.

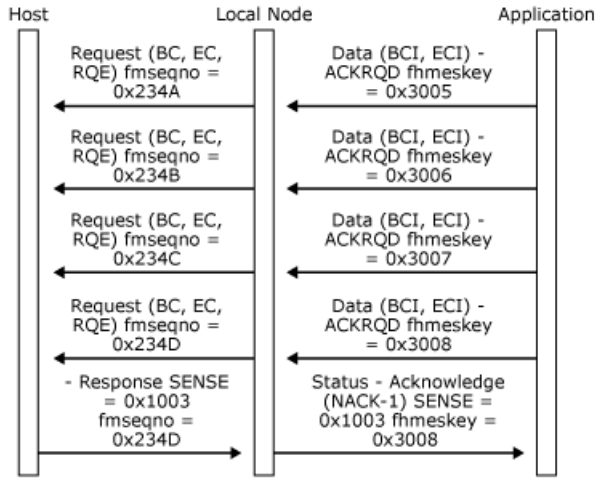
In the first figure, an application sets the **ACKRQD** field in an inbound data chain to get the host to confirm receipt of the chain and all previously sent chains.

Application sets ACKRQD field



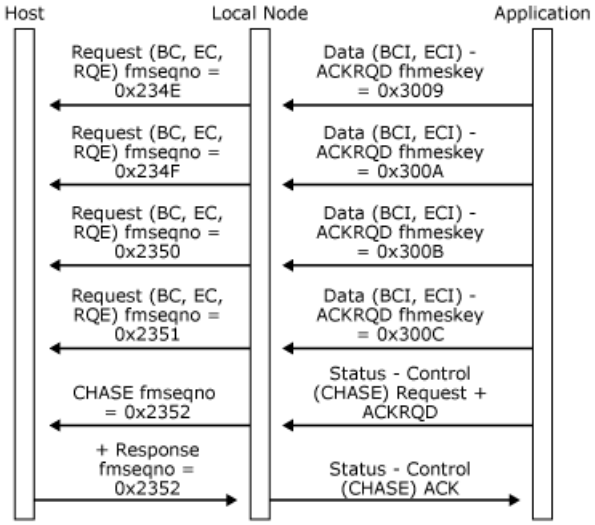
In the following figure, the **Status-Acknowledge(Nack-1)** rejects the last chain, but confirms receipt by the host of all previously sent data chains.

Status-Acknowledge(Nack-1) rejects the last chain, but confirms receipt



In the following figure, the application uses a **Status-Control(CHASE)** to get the host to confirm receipt of the corresponding **CHASE** request and all previously sent chains.

Using a Status-Control(CHASE) to get the host to confirm receipt of the corresponding CHASE request



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Confirmation and Rejection of Outbound Data

The local node sends chains of data from the host to the application with their **ACKRQD** field set as follows:

- **ACKRQD** set

  If the corresponding SNA request was received specifying definite response, and the **BIND** parameters specify that the primary uses definite or definite/exception chain response mode.

1. **ACKRQD** not set, response mode

   If the corresponding SNA request was received specifying exception response, and the **BIND** parameters specify that the primary uses exception or definite/exception chain response mode.

2. **ACKRQD** not set, no-response mode

   If the corresponding SNA request was received specifying no response, and the **BIND** parameters specify that the primary uses no-response chain response mode.

In case 1, the application should always send an acknowledgment as follows:

- If the application accepts the data, it should return a Status-Acknowledge(Ack) message.
- If the application wants to reject the data, it should return a **Status-Acknowledge(Nack-1)** message carrying the appropriate SNA sense codes.

In case 2, the application should only send an acknowledgment in the following cases:

- If the application wants to reject the data, it should return a **Status-Acknowledge(Nack-1)** message carrying the appropriate SNA sense codes.
- The application can send a courtesy acknowledgement to a Request Exception (RQE) message to clear up correlation data within the local node. (For more information, see Outbound Data.)

In case 3, the application should not send acknowledgments. However, the sending of a **Status-Acknowledge(Ack)** or **Status-Acknowledge(Nack-1)** by the application has no effect. It is discarded.

Whenever an application sends a **Status-Acknowledge(Ack)** or **Status-Acknowledge(Nack-1)** to a received Data message, it implicitly confirms receipt of this and all previously received Data messages.

In case 2, the host can issue a **CHASE** request. The local node sends a **Status-Control(CHASE) Request** with **ACKRQD** set to the application. When the application is in a position to confirm receipt of all outstanding data, it should issue a **Status-Control(CHASE) Acknowledge** message, which the local node converts into a positive response to **CHASE** for the host.

In cases 1 and 2, if the local node detects an error in a received request, it converts the request into a special Data message, which it passes to the application. Regardless of the chain response mode specified for the secondary in the **BIND** parameters, this **Data** message has the following characteristics:

- **ACKRQD** is set. The application must confirm receipt using a Status-Acknowledge(Ack) message.
- The Sense Data Indicator (SDI) application flag is set to indicate that this is a special Data message used to inform the application of an error detected by the local node.
- The End Chain Indicator (ECI) application flag is set to indicate that the received chain has now terminated.
- The first four bytes of the buffer element hold the SNA sense codes detected by the local node that caused the termination.

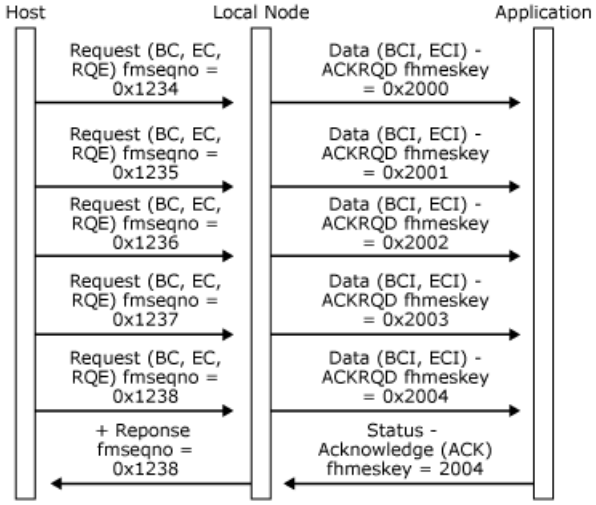
This mechanism enables:

- The application to reject any previously received **Data** messages.
- The local node to inform the application of any errors it detects in received requests.
- The local node to send negative responses in the correct order.

The following three figures illustrate the outbound data confirmation and rejection protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

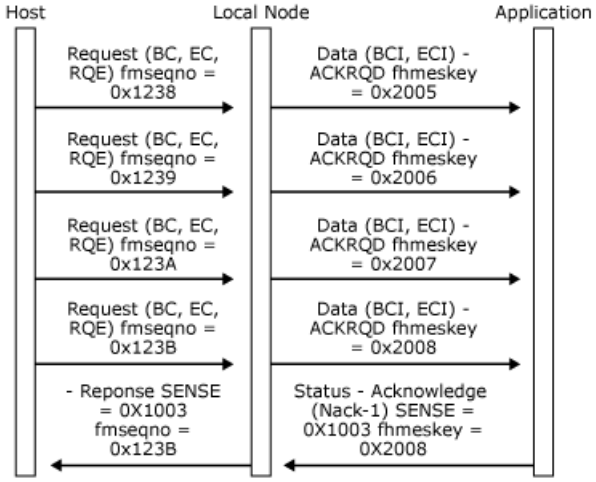
In the first figure, the host sends a definite response chain to get the application to confirm receipt of the RQD request and all previously sent RQE chains.

Host sends a definite response chain



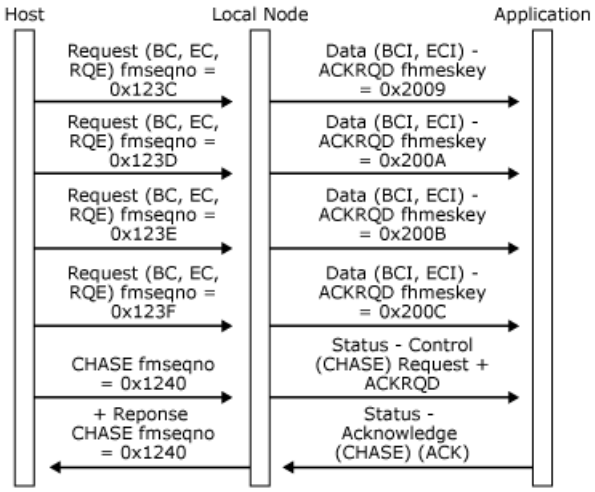
In the following figure, a **Status-Acknowledge(Nack-1)** from the application rejects the last chain and confirms receipt of all previously sent data chains.

Status-Acknowledge(Nack-1) rejects the last chain and confirms receipt



In the following figure, the host sends a **CHASE** request to get the application to confirm receipt of the **CHASE** and all previously sent chains.

Host sends a CHASE request



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Shutdown and Quiesce

Both shutdown and quiesce protocols involve a half-session entering a quiesced state, in which it cannot send any more normal flow requests, but must continue to receive and respond to requests from its session partner. The essential differences are that shutdown can only be initiated by the host and only requires that the secondary quiesce as soon as is convenient (usually at the end of a bracket). Quiesce can be initiated by both the host and the application and requires that the recipient quiesce at the end of the chain.

If the application has been quiesced but still attempts to send inbound Data messages, they will be rejected with Status-Acknowledge(Nack-2) messages. The application can, however, continue to generate status messages.

In This Section

- Shutdown
- Quiesce

Shutdown

The shutdown protocol provides a means for the host application to stop the application from sending any further normal flow requests. This protocol is used when the host application wants to end the session in an orderly manner and is only available for sessions using function management (FM) profile 3 or 4.

If the local node receives a **SHUTD** request from the host, it issues a **Status-Control(SHUTD) Request** (without **ACKRQD**) to request the application to enter a quiesced state at a convenient time. The application determines what is convenient. For example, it could be after a **Status-Session(BETB)** has been received.

When the application decides it is ready to quiesce, it should issue a **Status-Control(SHUTC) Request** (again without **ACKRQD**) to indicate this transition. The local node will notify the host of this change by sending a **SHUTC** request. The host can continue sending normal flow outbound requests and can subsequently take one of the following actions:

- The host terminates the primary logical unit (PLU) session by sending an **UNBIND** request. The local node closes the PLU connection by sending a Close(PLU) Request to the application. The system services control point (SSCP) session remains active.

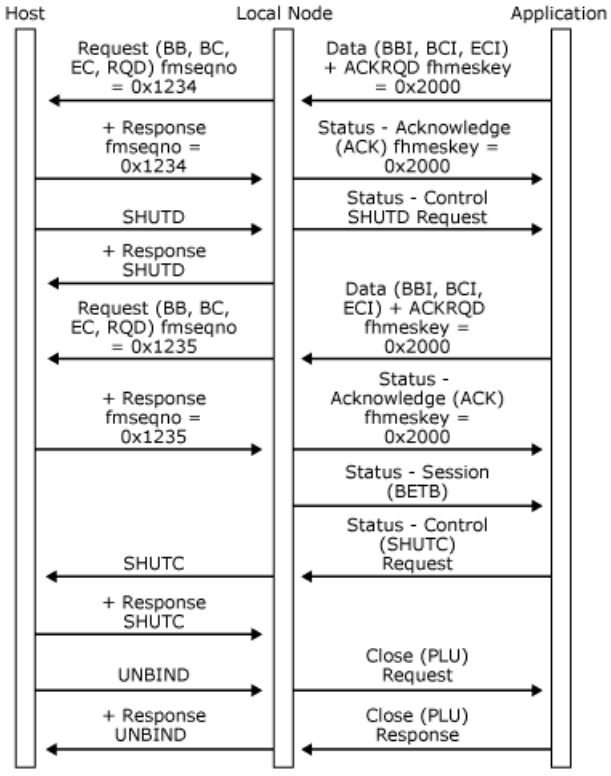
- The host abandons the shutdown procedure by sending an **RELQ** request. The local node sends a **Status-Control(RELQ) Request** (with **ACKRQD**) to the application to indicate that it can now resume sending on the PLU session. **RELQ** is only supported on sessions using FM profile 4.

- The host resets the session by sending **CLEAR**, a Transmission Service profile (TS profile) 3 or 4. One of the effects of this is to release the quiesced state. (For more information, see Recovery.)

The following two figures illustrate the shutdown protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

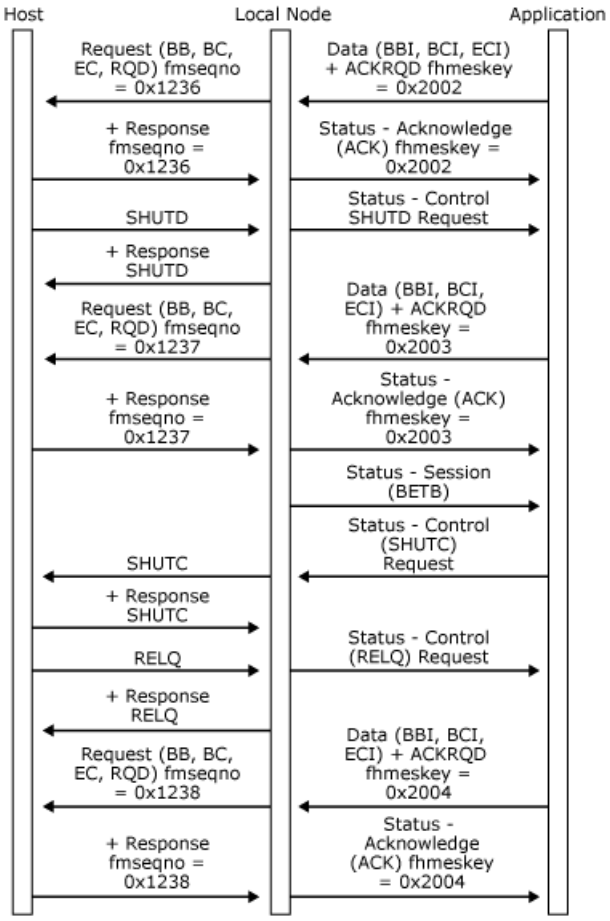
In the following figure, the host sends **SHUTD** while the application is sending in the in-bracket state. The application completes the bracket, sends **Status-Control(SHUTC) Request**, and the host terminates the PLU session by sending **UNBIND**. The local node closes the PLU connection.

Host sends SHUTD while the application is sending in the in-bracket state



In the following figure, the host sends **SHUTD** while the application is sending in the in-bracket state. The application completes the bracket, sends **Status-Control(SHUTC) Request**, and then the host releases the application from the quiesced state by sending **RELQ**. The local node sends a **Status-Control(RELQ) Request** to the application, which initiates a new bracket.

Host sends SHUTD while the application is sending in the in-bracket state



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Quiesce

The quiesce protocol is only supported on sessions using function management (FM) profile 4. The quiesce protocol can be initiated by either half-session.

When an application wants to quiesce its partner half-session in the host, it sends a **Status-Control(QEC) Request** to the local node. The node generates a **QEC** request to the host, which asks the host to quiesce after completing the current outbound chain.

If the host quiesces, it sends a **QC** request, which the local node presents to the application as a **Status-Control(QC) Request** (with **ACKRQD**). The host remains in a quiesced state until the application sends a **Status-Control(RELQ) Request**. The local node sends the **RELQ** request to the host, and the host resumes communications on the primary logical unit (PLU) session.

If the attempt to quiesce the host fails, the host responds with a negative **QEC** response, which the local node presents to the application as a **Status-Control(QEC) Negative-Acknowledge-1**.

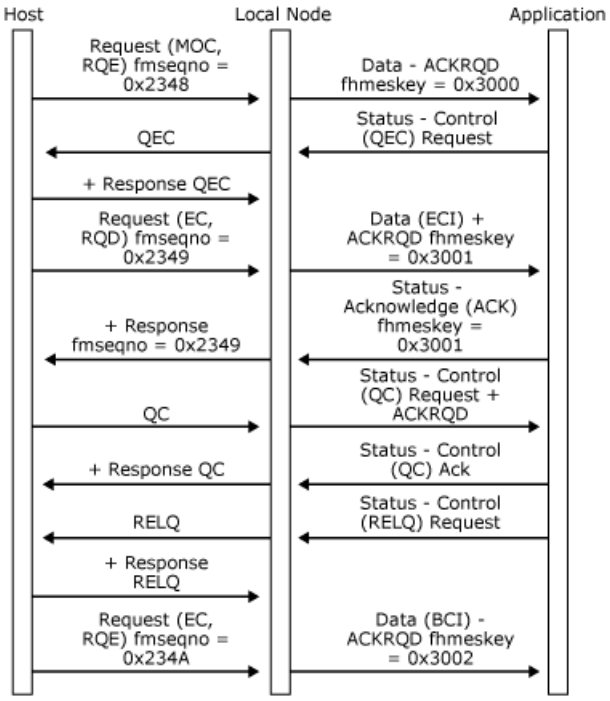
Conversely, a **Status-Control(QEC) Request** (without **ACKRQD**) is presented to the application if a **QEC** request is received from the host. In this direction, **QEC** cannot be rejected. The local node will always force the application to quiesce after presenting it with a **Status-Control(QEC) Request** by rejecting further attempts to send inbound data. When the application has quiesced, it should send a **Status-Control(QC) Request** to the local node, which sends a **QC** request to the host. The application can subsequently be released by an **RELQ** request from the host, which the local node presents to the application as a **Status-Control(RELQ) Request**.

The receipt of a **CLEAR** or **UNBIND–BIND** sequence, **Close(PLU)–Open(PLU)**, causes the quiesced state to be released.

The following three figures illustrate the quiesce protocols between the local node and the application and how those protocols relate to the underlying SNA protocols.

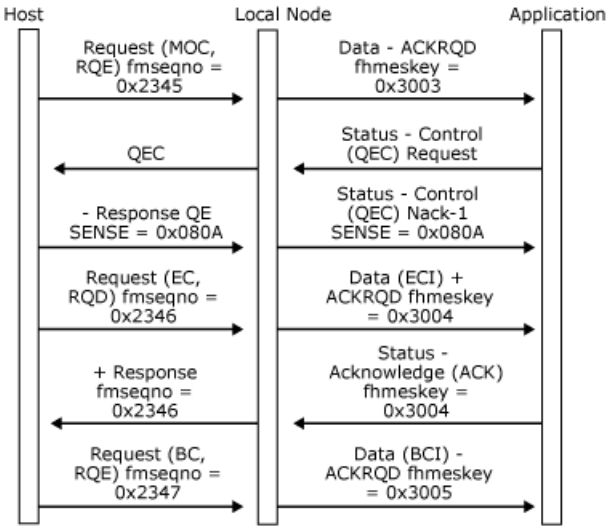
In the first figure, the application quiesces the host and then releases the quiesce.

Application quiesces the host and releases the quiesce



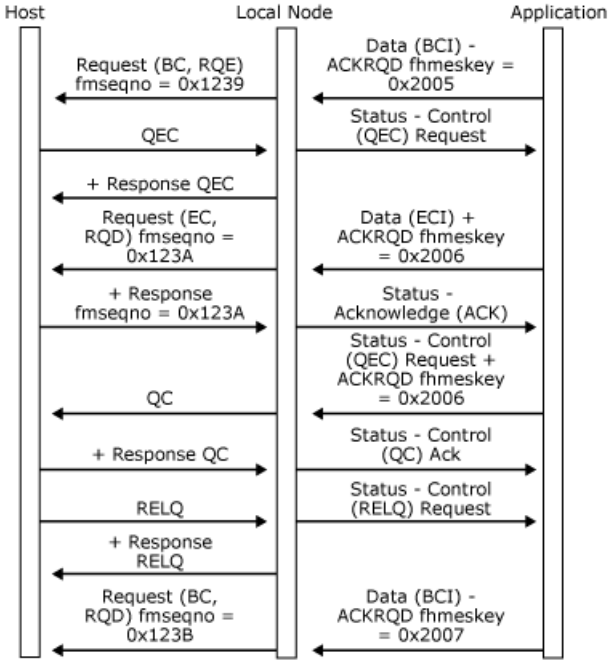
In the following figure, the application attempts to quiesce the host, but the host rejects the quiesce and continues with the next chain.

Application attempts to quiesce the host, but the host rejects and continues with the next chain



In the following figure, the host sends **QEC** while the application is sending a chain. The application completes the chain and sends a **Status-Control(QC) Request**. The host releases the quiesce by sending **RELQ**, and the local node sends a **Status-Control(RELQ) Request** to the application, which then initiates a new chain.

Host sends QEC while the application is sending a chain



See Also

Reference

LUSTATs

Concepts

Outbound Chaining

Inbound Chaining

Segment Delivery

Application-Initiated Termination

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

Recovery

This section covers a variety of issues pertaining to error recovery.

In This Section

Application CANCEL

One of the parameters on the Open(PLU) OK Response, which the application sends to the local node, specifies whether the application generates **CANCEL** (or **EC**) to terminate an inbound chain that has received a negative response. If this option is not selected, the local node generates a **CANCEL** request when it receives a negative response from the host to an incomplete chain.

See Also

Reference

RQR and CLEAR

STSN

Concepts

Direction after Receiving a Negative Response

Direction after Sending a Negative Response

Critical Failure

Link Service Failure

Local Node Failure

Client Failure

Direction after Receiving a Negative Response

Within the local node, error recovery for a half-duplex application (as specified by byte 7 bit 2 of the **BIND**) is assumed to be the responsibility of the host. However, the application must be aware that an error recovery state has been entered to obey the direction protocol.

When an application using half-duplex protocols (flip-flop or contention) receives a negative response to an inbound chain that it sent that does not refer to a race, it must assume receive state. The sense codes used for race conditions that do not require the transition to receive state are listed in the following table.

Sense code	Description
0x080B	Bracket race error
0x081B	Receiver in transmit mode

See Also

Reference

RQR and CLEAR

STSN

Concepts

Application CANCEL

Direction after Sending a Negative Response

Critical Failure

Link Service Failure

Local Node Failure

Client Failure

Direction after Sending a Negative Response

When an application using half-duplex flip-flop protocol sends a negative response to an outbound chain (or sends a Status-Acknowledge(Ack) to a **DATAFMI** message with **SDI** set) that does not refer to a race, the application must assume an error recovery pending state. The sense codes used for race conditions that do not require the transition to error recovery pending state are listed in the following table.

Sense code	Description
0x080B	Bracket race error
0x0813	Bracket bid reject (no RTR forthcoming)
0x0814	Bracket bid reject (RTR forthcoming)
0x081B	Receiver in transmit mode

The application must therefore examine the sense code on an **SDI** message to detect such races.

Error recovery pending state differs from receive state only in one respect: The application can convey sense information to the host using **Status-Control(LUSTAT).** (For more information, see LUSTATs.) The **LUSTAT** must not have the change direction (CD) or end bracket (EB) flags set. (The host already has direction, and the bracket must not be terminated prematurely by the application.) Host Integration Server also enables the function management interface (FMI) application to send **Status-Control(LUSTAT)** in receive state.

An application using the half-duplex contention protocol does not have an error recovery pending state, and must enter contention state whenever it sends a negative response.

Note
If the chain is canceled by the host with CD on the **CANCEL** , the application must assume send state.

See Also

Reference

RQR and CLEAR

STSN

Concepts

Application CANCEL

Direction after Receiving a Negative Response

Critical Failure

Link Service Failure

Local Node Failure

Client Failure

Critical Failure

When an application makes a protocol error in sending data, the local node rejects the data using a Status-Acknowledge(Nack-2) with a sense code indicating the reason for failure. This message has a critical failure flag that indicates whether the local node has marked the session as unrecoverable. The sense codes are listed in FMI Status, Error, and Sense Codes.

If the error is noncritical, the application can proceed as if the message that caused the error had not been sent. If the error is critical, the local node issues a Close(PLU) Request to the application (providing that the primary logical unit (PLU) connection is open), which means that the application cannot communicate on the PLU-SLU session until an **UNBIND–BIND** sequence is received from the host. The local node also sends a **TERM-SELF** request to the host to elicit an **UNBIND**. Therefore, the application does not need to issue a **LOGOFF** request on the system services control point (SSCP) session.

See Also

Reference

RQR and CLEAR

STSN

Concepts

Application CANCEL

Direction after Receiving a Negative Response

Direction after Sending a Negative Response

Link Service Failure

Local Node Failure

Client Failure

RQR and CLEAR

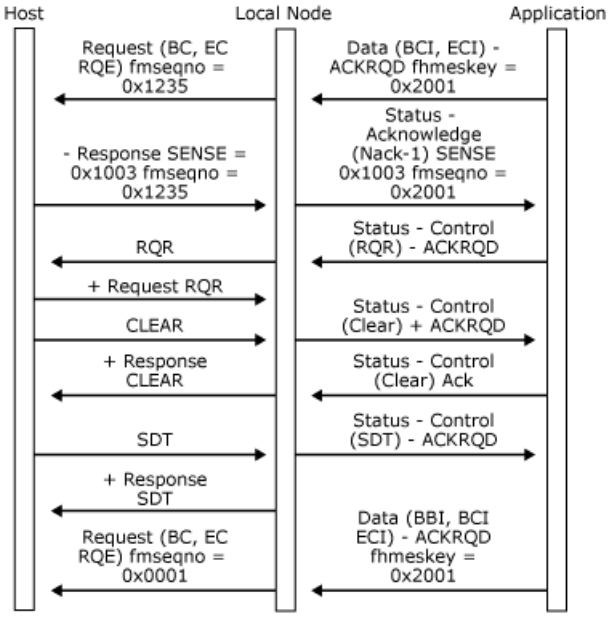
An application using Transmission Service profile (TS profile) 4 can request the session to be recovered by sending **Status-Control(RQR)**. The local node presents this to the host as an **RQR** request. Note that, if the application has received a critical Status-Acknowledge(Nack-2), this option cannot be taken because the local node will send a Close(PLU) Request immediately following the **Status-Acknowledge(Nack-2)** to the application, and the primary logical unit (PLU) connection will no longer be valid. The **RQR** message requests the host to reset the session by sending a **CLEAR** request, as shown in the following figure.

The receipt of **CLEAR** causes the application to reset its session state to that following the **BIND**, the **Open(PLU)**.

Another way for the application to deal with error conditions is to ask for an **UNBIND** by sending **Status-Control(RSHUTD)**. (For more information, see Application-Initiated Termination.) Note that this may not require the host to supply a new **BIND**, depending on the host configuration. A new SSCP request may be required (such as **LOGON**).

In the following figure, the application requests recovery by issuing **Status-Control(RQR)**. The host sends **CLEAR**, and the application must reset its session to state that it was following the **BIND (Open(PLU))**. In this case, the application is now between brackets and awaiting start data traffic (SDT).

Application requests recovery by issuing Status-Control(RQR)



See Also

Reference

STSN

Concepts

Application CANCEL
Direction after Receiving a Negative Response
Direction after Sending a Negative Response
Critical Failure
Link Service Failure
Local Node Failure
Client Failure

STSN

Set and test sequence numbers (STSN) are used on sessions with Transmission Service profile (TS profile) 4 for applications to maintain transaction processing sequence numbers between sessions. This enables both partners on the session to discover the amount of data lost after a **CLEAR** or **UNBIND–BIND** sequence.

The **STSN** message is the only one that can reset such transaction processing sequence numbers. **BIND**, **UNBIND**, and **CLEAR** do not affect them.

If the application wants to maintain such transaction numbers, it must specify the **APPLTRAN** option in the Open(PLU) OK Response. The host can send **STSN** after a **BIND** or **CLEAR** before sending **SDT** to set or test the application's transaction numbers. The local node resets its internal session sequence numbers to zero on receipt of **BIND** or **CLEAR**. When the local node receives an **STSN** specifying **SET** (or **SET** and **TEST**) for one half-session, it resets the corresponding internal session sequence number.

Unless both half-session actions are ignore (the action byte is 0x00), the **STSN** request is passed to the application (provided that it specified **APPLTRAN**), with the action byte and the two sequence numbers from the request, as a **Status-Control(STSN)**. (For more information, see Status-Resource.) The application must examine the action byte to determine whether the action is ignore, set, test, or set and test. The application must send a positive response (**Status-Control(STSN) Acknowledge**) to the **STSN**, with the sensed sequence numbers if required (sense or set and test). The local node is responsible for generating the correct result code for the **STSN RSP**.

Note that the application should perform the sense part of **STSN** first (by examining bits 0 and 2 of the action byte for the secondary-to-primary flow and primary-to-secondary flow respectively). The set part of the **STSN** is then performed (by examining bits 1 and 3 of the action byte).

The application should increment its transaction numbers when sending and receiving normal flow request/response units (RUs) from the host. (Note that **Status-Control** messages corresponding to normal flow data flow control (DFC) requests cause the transaction numbers to be incremented.) The sequence number is reported on **DATAFMI** messages and **Status-Acknowledge** messages. The application should be aware that, if a message from the host fails receive checks (and is converted to an **SDI** message), sub-network access protocol (SNAP)-2.1 will purge the remainder of the chain from the host, and the application may miss some sequence numbers. Therefore, the application should reset its primary-to-secondary transaction number from the next outbound data after processing an **SDI** message.

Note that the second application flag byte is not valid for **Status-Control(STSN)**. It is used for the **STSN** control byte.

See Also
Reference
RQR and CLEAR
Concepts
Application CANCEL
Direction after Receiving a Negative Response
Direction after Sending a Negative Response
Critical Failure
Link Service Failure
Local Node Failure
Client Failure

Link Service Failure

When the server running a link service fails, the local node is informed of this. It treats the problem as a link outage with outage code 0x0D. This is reported to any active 3270 emulation sessions as a communications check code (–+z_nnn). The local node will attempt periodically to reconnect to the link service.

See Also

Reference

RQR and CLEAR

STSN

Concepts

Application CANCEL

Direction after Receiving a Negative Response

Direction after Sending a Negative Response

Critical Failure

Local Node Failure

Client Failure

Local Node Failure

If the local node fails, applications are informed of this by the path error return code from the Dynamic Access Module (DMOD) on the sbpurcvx call, or from the routing procedure. All connections that use the destination locality value for which the path error is reported are closed. The application must do the following:

- Clean up resources related to the closed connections, including resetting presentation spaces and displaying a communications check code

  (–+z_nnn) on the status line.

- Attempt to reestablish connection with a local node by reinitiating the resource location process.

See Also
Reference
RQR and CLEAR
STSN
Concepts
Application CANCEL
Direction after Receiving a Negative Response
Direction after Sending a Negative Response
Critical Failure
Link Service Failure
Client Failure

Client Failure

If the client computer fails, the local node terminates the application's PLU-SLU session (if it is active) by sending **TERM-SELF**. The system services control point (SSCP) and primary logical unit (PLU) connections are both marked as closed and cannot be reused without being reopened. Internally, the local node treats such a failure in the same way as the receipt of a Close(SSCP) Request from the application.

See Also

Reference

RQR and CLEAR

STSN

Concepts

Application CANCEL

Direction after Receiving a Negative Response

Direction after Sending a Negative Response

Critical Failure

Link Service Failure

Local Node Failure

Application-Initiated Termination

An application on a session with function management (FM) profile 3 or 4 can request termination of the primary logical unit (PLU) session. It should only do so if it has previously ensured that it is in a state where the PLU session can be terminated, that is, between-chain and between-bracket. Terminating the PLU session does not affect the state of the system services control point (SSCP) session.

Note that an application can issue a character coded or field formatted **LOGOFF** command on the SSCP session or send a Close(PLU) Request to get the local node to send **TERM-SELF** on the application's behalf. All of these will elicit an **UNBIND**, either immediately or after session clean-up in the host.

The application requests termination of the PLU session by sending a **Status-Control(RSHUTD) Request** to the local node, which generates an SNA **RSHUTD** request to the host.

After sending the **Status-Control(RSHUTD) Request**, the application must remain capable of accepting and responding to all outbound data it receives. The application can now expect one of two messages, depending on whether the state of the PLU session allows it to be terminated and whether the host wants to terminate the PLU session:

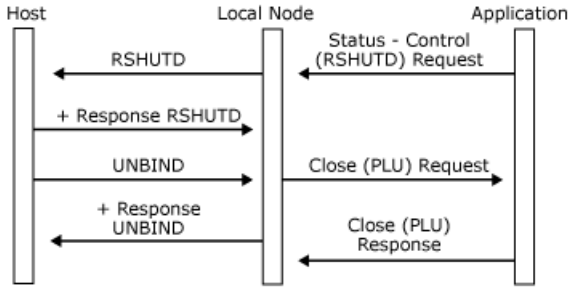
- If the state of the PLU session allows it to be terminated, and the host wants to terminate the PLU session, the host generates a positive response to the **RSHUTD** request, which can be followed by an **UNBIND** request. The local node closes the PLU connection. For more information, see Closing the PLU Connection.

- If the state of the PLU session does not allow it to be terminated (for example, if the session is in-bracket), or the host does not want to terminate the PLU session at this time, the host generates a negative response to the **RSHUTD** request, which the local node presents to the application as a **Status-Control(RSHUTD) Negative-Acknowledge-1** carrying the sense codes supplied on the negative response. This indicates that the request to terminate the PLU session has been rejected by the host, and communication on the PLU session continues unaffected.

The following two figures illustrate the application-initiated termination protocol between the local node and the application and how this protocol relates to the underlying SNA protocols.

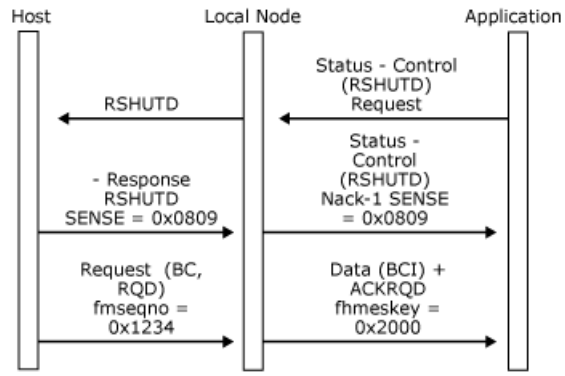
In the first figure, the application requests termination of the PLU session, and the host sends **UNBIND**. The local node closes the PLU connection.

Application requests termination of the PLU session, and the host sends UNBIND



In the following figure, the application requests termination of the PLU session, but the session is not in an appropriate state. The host sends a negative response to the **RSHUTD** request, which the local node presents as **Status-Control(RSHUTD) Negative-Acknowledge-1**. Communication continues on the PLU session.

Application requests termination of the PLU session, but the session is not in an appropriate state



See Also

Reference

LUSTATs

Concepts

Closing the PLU Connection

Outbound Chaining

Inbound Chaining

Segment Delivery

Response Time Monitor Data

Other Resources

Opening the PLU Connection

Using the PLU Session

Brackets

Direction

Pacing and Chunking

Confirmation and Rejection of Data

Shutdown and Quiesce

Recovery

LUSTATs

The data flow control (DFC) logical unit status **LUSTAT** message is used within SNA to convey four bytes of sense data to the other session partner. It can also be used simply to send a response header (RH) to the other session partner (for example, to open a bracket). (For more information, see the figures in Bracket Initiation.) The message flows on the normal flow and so is subject to direction restrictions. However, it can be sent without end bracket (EB) or change direction (CD) on a half-duplex flip-flop session that is in error recovery pending state. (For more information, see Recovery.)

The local node allows the application to send **Status-Control(LUSTAT) Request** messages at any time that data traffic is active, except while sending data in chain. If the application is in a receiving state (using half-duplex protocol), the **LUSTAT** is queued up and used to provide the sense codes, which are filled into the next outbound request, and the SDI flag is set. The application can therefore present the sense codes for an error state without waiting for the next outbound data if required.

The first byte of sense data must be 0x08 to generate a **DATAFMI** message with **SDI** (to be converted to a negative response). Other **LUSTATs** are left queued on the session until they can be sent.

If multiple **Status-Control(LUSTAT)** messages are sent by the application while in a receive state, the local node queues them all. When outbound data has been delivered to the application with **SDI** set, as indicated earlier, and the application has converted it to a Status-Acknowledge(Ack), the local node sends the negative response and the remaining **LUSTATs** (which can now flow because the half-duplex flip-flop state is error recovery pending).

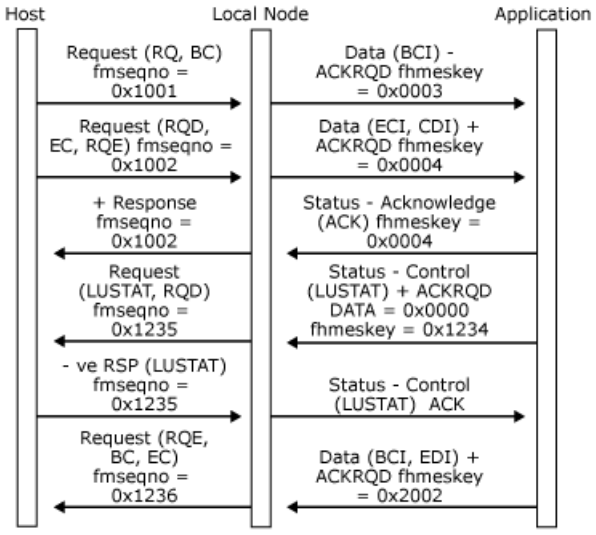
If the application intends to send multiple **Status-Control(LUSTAT)** messages to the host, it is possible that the host will attempt to initiate recovery before the last **LUSTAT** has been sent. In this case, the error recovery chain will be rejected by the next **LUSTAT**.

Note that the application can send **Status-Control(LUSTAT) Request** with or without **ACKRQD**. The local node will map these to **RQD** and **RQE LUSTATs** respectively.

The following three figures illustrate the use of **Status-Control(LUSTAT)** messages by an application using the half-duplex flip-flop mode.

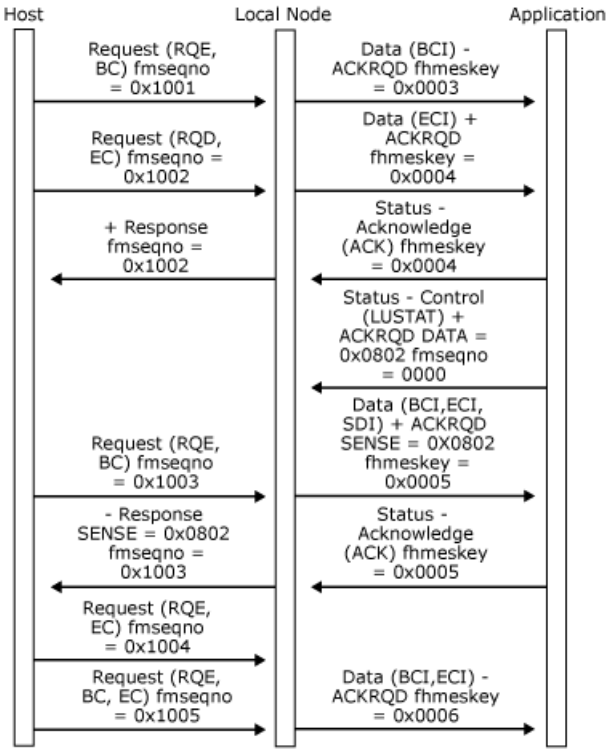
In the first figure, the application issues **Status-Control(LUSTAT)** when it has direction.

Application issues Status-Control(LUSTAT) when it has direction



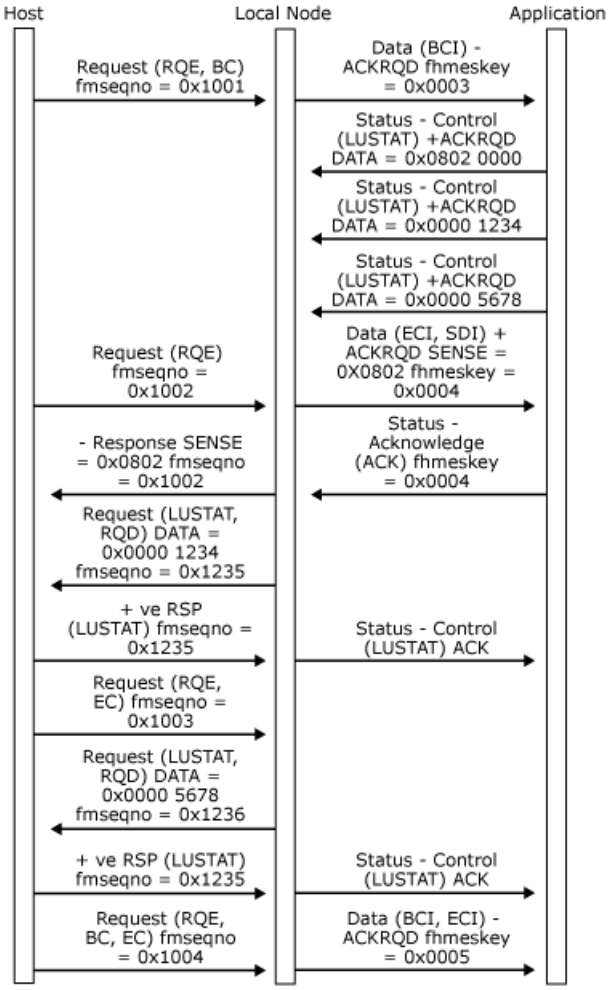
In the following figure, the application sends **Status-Control(LUSTAT)** request when receiving data between chain. Next, outbound data is delivered with **SDI** set, which gets converted to negative **RSP**.

Application issues Status-Control(LUSTAT) request when receiving data between chain



In the following figure, the application sends several **Status-Control(LUSTAT)** requests when receiving data in chain. Next, outbound data is delivered with **SDI** set which gets converted to negative response. Subsequent **LUSTATs** are sent to host.

Application sends several Status-Control(LUSTAT) requests when receiving data in chain



See Also

Concepts

Closing the PLU Connection

Outbound Chaining

Inbound Chaining

Segment Delivery
Application-Initiated Termination
Response Time Monitor Data
Other Resources
Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Response Time Monitor Data

For a 3270 display application, the local node maintains statistics on host response times—the time it takes the host to respond after the 3270 user presses ENTER or an AID key to send data to the host. These statistics can then be sent to the host for analysis.

The Status-RTM message, sent by the local node to the application, informs the application of the Response Time Monitor (RTM) parameters specified by the host. (For more information, see RTM Parameters.) These parameters specify whether RTM data is to be collected, whether the application is permitted to display RTM statistics locally, the time boundaries by which response times are to be grouped, and the definition of response time. The time can be measured until the first character of the host response reaches the screen, until the keyboard is unlocked, or until the user can send further data (change direction (CD) or end bracket (EB) received by the application).

If the host specifies that response times are to be measured for this session, the application is responsible for measuring response times and for reporting them to the local node. This involves:

- Starting a timer when the user presses the ENTER key or an AID key to send data to the host.
- Stopping the timer when the host's response to the inbound data is received, as defined by the RTM definition specified on the **Status-RTM** message.
- Reporting the response time to the host on the Status-Acknowledge(Ack) message, which acknowledges the host's response. One of the fields on this message specifies the last response time measured by the application, or specifies that no response time is to be reported.
- Optionally displaying the most recent response time as a last transaction time indicator (LTTI) on the 3270 emulation status line.

If the application wants to provide a local display of RTM data, it is responsible for maintaining its own response time statistics. It should use the same definition and boundaries as those specified on the Status-RTM message to ensure that the local data matches the data sent to the host by the local node. Note that the **Status-RTM** message can indicate that a local display of response times is not permitted. In this case, the application should not display either the response times or the LTTI.

See Also
Reference
LUSTATs
Concepts
Closing the PLU Connection
Outbound Chaining
Inbound Chaining
Segment Delivery
Application-Initiated Termination
Other Resources
Opening the PLU Connection
Using the PLU Session
Brackets
Direction
Pacing and Chunking
Confirmation and Rejection of Data
Shutdown and Quiesce
Recovery

Data Flow

The following topics describe data flows between the application and the local node.

In This Section

- Outbound Data
- Inbound Data
- Inbound Data from LUA Applications

Outbound Data

This section describes the outbound data flows from the local node to the application. The overall structure of the protocols described applies to the system services control point (SSCP) and primary logical unit (PLU) connections, but certain features (such as the use of delayed request mode) are only applicable to the PLU connection.

The local node presents data originating at the host to the application on different connections, depending on the SNA session on which the data flows, as follows:

- Function management data network services (FMD NS) (session services) data and function management data (FMD) originating at the host SSCP and directed to the Host Integration Server logical unit (LU) is sent to the application on the SSCP connection.

- FMD data originating at the host PLU and directed to an SNA server LU is sent to the application on the PLU connection.

For all connections, only FMD requests are presented to the application as Data messages (with message-type = DATAFMI). DFC and session control requests are used to generate **Status-Control** messages. (For more information, see Status-Control Message.)

The local node performs the data flow control state changes required by the response header (RH) indicators in the request, before sending a **Data** message to the application.

The SNA request transmission header (TH) and RH indicators are not available to the application on outbound **Data** messages. Instead, the local node provides application flags in the **Data** message header that reflect the settings of a subset of the RH indicators, but are interpreted by the local node to shield the application from the more obscure aspects of chaining and bracket usage. For a description of the available flags and the way in which the local node uses them on outbound data, see Application Flags.

For outbound data, the first byte is RU[0] for standard function management interface (FMI), and TH[0] for the logical unit application (LUA) variant of FMI.

All Data messages from the local node to an application contain a message key. The local node maintains a unique message key sequence for each outbound data flow to an application. When the local node sends a **Data** message to an application on a particular connection, it places the next message key in the outbound sequence into the message header, sets the application flags, and sends the message to the application. This means that the message key uniquely identifies a **Data** message on a particular connection between the local node and the application. Note that the local node also places message keys on outbound **Status-Control Request** messages.

The acknowledgment protocol enforced by Host Integration Server reflects the chain response protocol and request mode in use on the SNA session, as follows:

- Outbound **RQD** requests generate Data messages with **ACKRQD** set in the message header.

- Outbound **RQE** requests generate **Data** messages without **ACKRQD** set.

- Outbound **RQN** requests generate **Data** messages without **ACKRQD** set.

- If the session uses primary immediate request mode, a **Data** message with **ACKRQD** set must be acknowledged by the application before further **Data** messages will be received.

- If the session uses primary delayed request mode, a **Data** message with **ACKRQD** set need not be immediately acknowledged by the application. **Data** messages will continue to be received.

Note that Host Integration Server enforces the equivalent of immediate response mode for the outbound data acknowledgment protocol for all connections. The application must send acknowledgments in order.

If the local node sets the **ACKRQD** field in the message header of a Data message to the application, it indicates that an acknowledgment to this **Data** message is required. The application acknowledges an outbound **Data** message by sending a **Status-Acknowledge** message to the local node on the same connection, which contains the same message key and sequence number fields as the **Data** message.

On receipt of a Status-Acknowledge(Ack), the local node correlates the message key with outstanding outbound messages and generates an SNA positive response to the appropriate SNA request.

The application should use the Status-Acknowledge(Nack-1) message as a negative acknowledgment. On receipt of a **Status-Acknowledge(Nack-1)**, the local node correlates the message with outstanding outbound messages and generates an SNA negative response plus sense data to the appropriate SNA request. The application supplies the sense data that should accompany the negative response as part of the **Status-Acknowledge(Nack-1)** message and must include the same message key, application flags, and sequence number fields as the **Data** message to which this is a negative acknowledgment.

Status-Control messages caused by expedited-flow requests can be sent at any time and do not affect the sending of positive or negative acknowledgment to normal flow outbound **Data** messages. The fact that they can occur between an outbound **Data** message and the matching **Status-Acknowledge** message is purely coincidental. For details about which **Status-Control** messages correspond to SNA requests, see Status-Control Message.

If errors are detected in the format of a normal flow request from the host or the request is inappropriate for the state of the session, the local node generates an error Data message for the application with the following characteristics:

- The SDI and ECI application flags are set.
- The sense code associated with the error occupies the first four bytes of the **Data** message. (For more information, see Status-Control Message.)
- **ACKRQD** is set.

The application should return a Status-Acknowledge(Ack), and the local node generates a negative response carrying the sense code appropriate to the detected error. This mechanism does the following:

- Informs the application of the detected error.
- Allows the application to respond to any previously received data before the local node sends the negative response to this Data message.

On sessions where the application is receiving a series of RQE chains, the local node will be retaining correlation information for each chain (in case the application wants to send negative responses to any of the chains). If the local node runs out of correlation table entries, it will attempt to allocate more entries and (if this fails) will be forced to terminate sessions. To prevent this, the application should provide **Status-Acknowledge(Ack)** messages for RQE data that it does not want to reject in this case. A response after five consecutive RQE chains should be sufficient. Such messages are referred to as courtesy acknowledgements and do not give rise to a response to the host, but merely free internal correlation data.

The following six figures illustrate the data acknowledgment protocol enforced between the local node and the application, and show the effects of the application generating positive and negative **Status-Acknowledg**e messages.

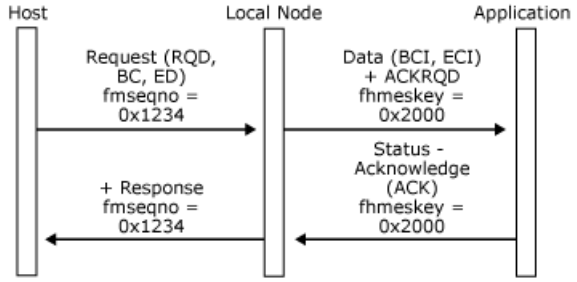
The figures show:

- The relevant RH flags in SNA requests/responses.
- The sequence number of SNA requests/responses.
- Any sense data (shown as "SENSE=...") on SNA requests/responses and **Status-Acknowledge** messages.
- The **ACKRQD** field in Data messages.
- The message key field in **Data** messages.

For simplicity, all messages are assumed to be FM data flowing on the same PLU session.

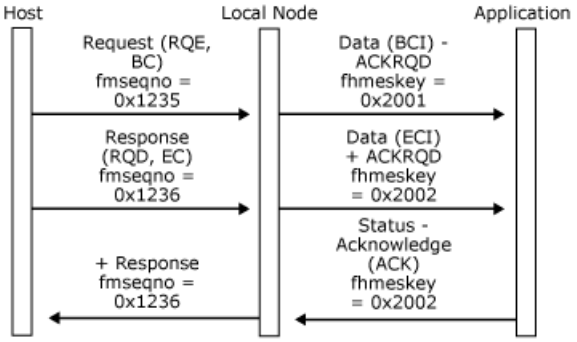
In the following figure, the application accepts a **Data** message corresponding to a definite-response RU.

Application sends a Data message corresponding to a definite-response RU



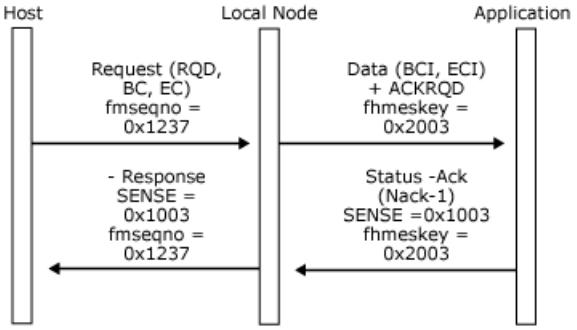
In the following figure, the application accepts a **Data** message corresponding to a multi-RU definite-response chain.

Application accepts a Data message corresponding to a multi-RU definite-response chain



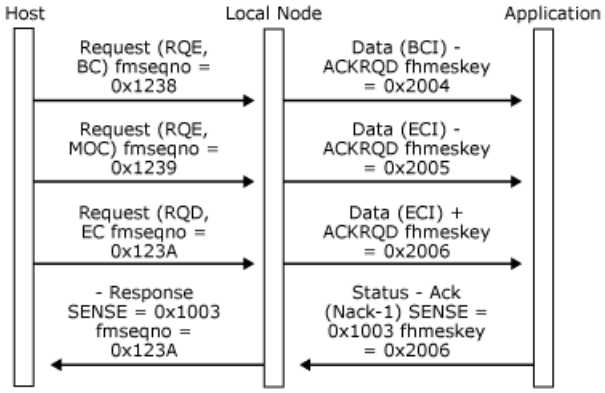
In the following figure, the application rejects a **Data** message corresponding to a definite-response chain.

Application rejects a Data message corresponding to a definite-response chain



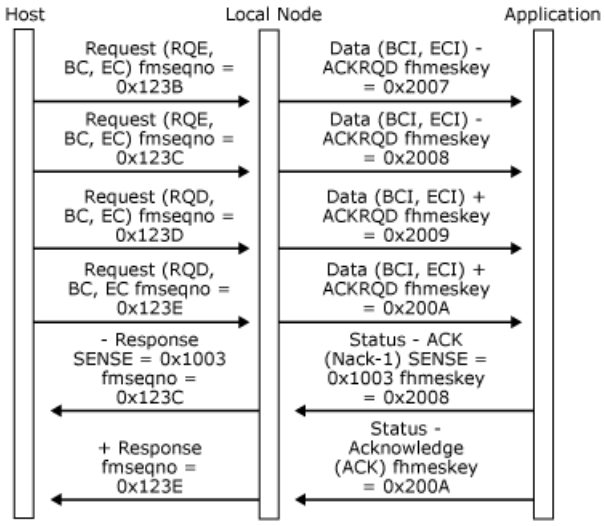
In the following figure, the application rejects a **Data** message corresponding to a multi-RU definite-response chain.

Application rejects a Data message corresponding to a multi-RU definite-response chain



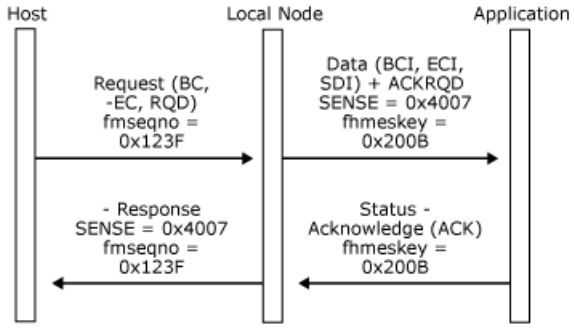
In the following figure, the local node enforces immediate response mode. Responses must be sent in sequence. The application rejects the second exception-response chain and accepts the definite-response chain, which implies acceptance of the third exception-response chain.

Local node enforces immediate response mode



In the following figure, the local node detects a chaining error (RQD but not EC) in data destined for the application. (This example requires the receive check 0x4007 to be in force. For more information, see Opening the SSCP Connection.)

Local node detects a chaining error in data destined for the application



See Also

Concepts

Inbound Data

Inbound Data from LUA Applications

Inbound Data

This section describes inbound data flows from the application to the local node. The overall structure of the protocols described applies to the system services control point (SSCP) and primary logical unit (PLU) connections, but more complex aspects (such as the use of delayed request mode) are only applicable to the PLU connection.

The application can send inbound data on any of the connections, as follows:

- Function management data network services (FMD NS) (session services) and function management data (FMD) character-coded data intended for the host SSCP should be sent to the local node on the SSCP connection.
- FMD data intended for the host PLU should be sent to the local node on the PLU connection.

The application cannot use Data messages to send data flow control (DFC) or session control request messages to the host. Instead it must use **Status-Control** messages. (For details, see Status-Control Message.)

For all connections, the application must fill in certain key fields in the **Data** message's header. In particular it must:

- Set the message-type to **DATAFMI**.
- Allocate a new message key for inbound **Data** messages on this connection.
- Set the **ACKRQD** field if required.
- Set the application flags. (For more information, see Application Flags.)

The **nxtqptr**, **hdreptr** and **numelts** fields in the message header, and the **elteptr** and **startd** fields in the message elements are set up by the Host Integration Server buffer management routines. (For more information, see DL-BASE/DMOD Interface.) The application is responsible for setting the **endd** field.

If the application does not have access to these routines (for example, when the operating environment does not support intertask procedure calls and shared memory), all the fields in the header must be set by the application.

The transmission header (TH) and response header (RH) indicators are not available to the application on inbound Data messages. The application should set the appropriate application flags in the message header to control chaining, direction, and so on. For a description of the available application flags for inbound data and later topics in this section for a description of how the flags are used to control inbound data flows, see Application Flags.

For inbound data, the first byte is RU[0] for standard function management interface (FMI).

The message key supplied by the application in the inbound **Data** message header is used by the local node to indicate which **Data** message on this connection an outbound **Status-Acknowledge** refers to. The application should maintain a unique message key sequence for the inbound data flow on each connection it has with the local node, so that the application can use the message key to correlate inbound **Data** messages and outbound **Status-Acknowledge** messages on the connection. Note that the application must also provide a message key on **Status-Control Request** messages to differentiate between multiple **RQE LUSTAT** messages.

The inbound data acknowledgment protocol reflects the secondary chain response protocol and request mode in use on the session, as follows:

- Inbound Data messages with **ACKRQD** set in the header generate **RQD** requests.
- Inbound **Data** messages without **ACKRQD** set in the header generate **RQE** or **RQN** requests depending on the chain response protocol.
- The application should only set **ACKRQD** on **Data** messages that have the end chain indicator (ECI) application flag set.
- If the session specifies that the secondary uses immediate request mode, the application can still send further **Data** messages after sending data with **ACKRQD** set, even though it has not received a **Status-Acknowledge** message for that **Data** message. The messages are queued within the local node and are progressively sent as positive responses are

received.

- If the session specifies that the secondary uses delayed request mode, after sending a **Data** message with **ACKRQD** set, the application can continue to send **Data** messages.

If the application sets the **ACKRQD** field in the message header of a Data message, it indicates that it requires an acknowledgment to this **Data** message. The local node acknowledges an inbound **Data** message by sending a **Status-Acknowledge** message to the application on the same connection and using the same message key as the **Data** message. (For an illustration, see the first figure at the end of this topic.)

The local node processes inbound **Data** messages from the application through its internal state computers, assigns the correct SNA sequence number or an identifier for this flow, and sends the data in a request to the host. The chain-response type of the request depends on whether **ACKRQD** was set in the **Data** message and the session parameters.

The local node maps a positive response from the host to a Status-Acknowledge(Ack) to the application. The application can use the message key in the **Status-Acknowledge** to correlate the acknowledgment with the original **Data** message. Therefore, receipt of a **Status-Acknowledge(Ack)** for a particular **Data** message implies that the local node has received a positive SNA response from the host to the inbound SNA request. (For an illustration, see the second figure at the end of this topic.)

Note that responses are absorbed on the SSCP-PU session.

Note that outbound Status-Acknowledge(Ack) messages contain application flags and a sequence number. The application flags reflect the RH indicators in the response. The sequence number is the SNA sequence number from the response, and provides a mechanism for applications using Transmission Service profile (TS profile) 4 to track the SNA secondary sequence number corresponding to a unit of work.

The local node maps a negative response from the host to a Status-Acknowledge(Nack-1) message to the application. The application can use the message key in the **Status-Acknowledge** to correlate the negative acknowledgment with the original **Data** message. The outbound **Status-Acknowledge(Nack-1)** message contains the SNA sense codes and sequence number from the negative response. (For an illustration, see the third and fourth figures at the end of this topic.)

If the local node detects an error in the format of an inbound **Data** message, or the **Data** message is not appropriate to the current state of the session, it sends a Status-Acknowledge(Nack-2) to the application containing an error code. (For a list of error codes, see Error and Sense Codes.) The local node does not send a request to the host corresponding to the **Data** message in error and does not advance the SNA sequence number for the session. The application can use any message key in its next inbound **Data** message (assuming the error does not cause a critical failure).

An example of a serious chaining error, where the application sends a **Data** message with **ACKRQD** but without ECI in the application flags, is shown in the last figure at the end of this topic. Note that after detecting this particular error, the local node marks the application's connection as critically failed, closes the connection, and sends a **TERM-SELF** request to the SSCP to elicit an **UNBIND**. (For more information, see Recovery.)

Inbound **Status-Control** messages, which cause the generation of expedited-flow requests, can be sent at any time and do not affect the sending of a positive or negative acknowledgment to inbound **Data** messages. For details about which **Status-Control** messages correspond to SNA expedited-flow requests, see Status-Control Message.

The following five figures illustrate examples of the inbound data acknowledgment protocols (and the underlying SNA protocols) for different chain-response types and secondary session request modes.

The figures show:

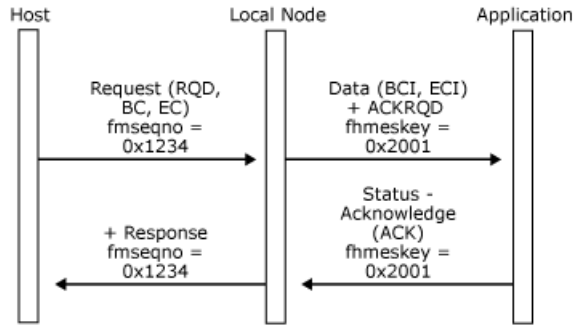
- The **ACKRQD** field on Data messages.
- The message key on **Data** messages.
- Any relevant application flags on **Data** messages.
- Error codes (shown as "ERROR=...") on **Data** messages.
- Relevant RH flags on SNA requests/responses.
- Sequence numbers on SNA requests/responses.

- Sense codes (shown as "SENSE=....") on SNA requests/responses.

For simplicity, all messages are assumed to be flowing on the same PLU session.

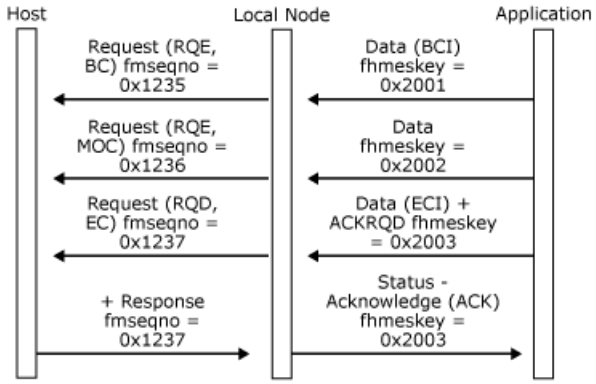
In the following figure, the application successfully sends a **Data** message.

Application successfully sends a Data message



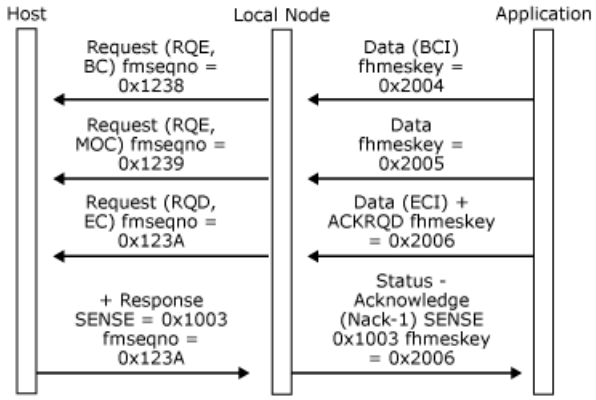
In the following figure, the application successfully sends a chain of **Data** messages.

Application successfully sends a chain of Data messages



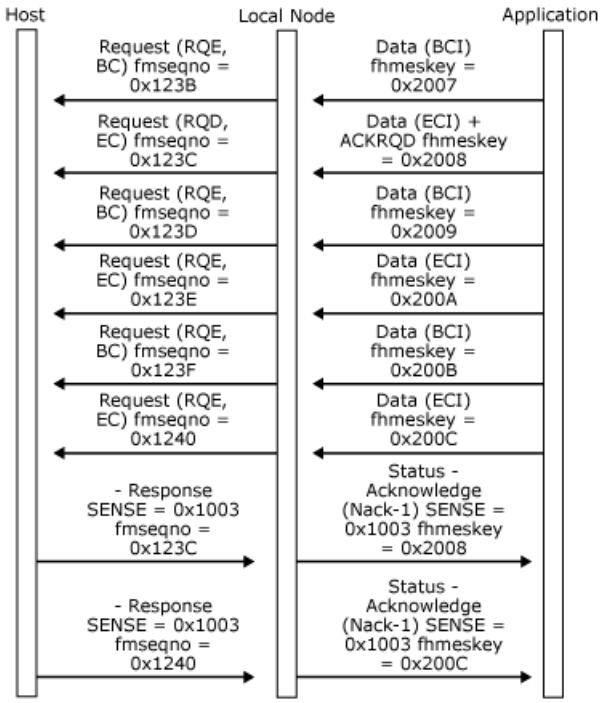
In the following figure, the host rejects a chain of **Data** messages.

Host rejects a chain of Data messages



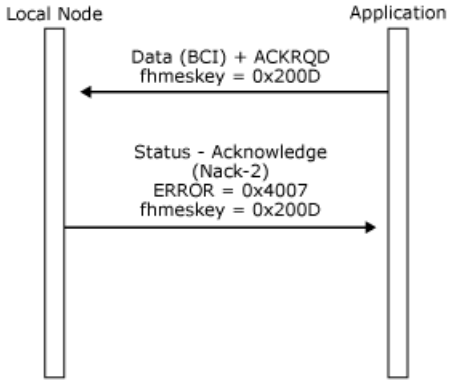
In the following figure, the host rejects the first definite-response chain and rejects the third exception-response chain on a delayed request session. Note that the negative response to the third chain implies a positive response to the second chain.

Host rejects the first definite-response chain



In the following figure, the local node detects the application's invalid use of **ACKRQD** without the ECI application flag on a **Data** message. Note that no data is sent to the host. However, because the error is critical, the local node will send a **TERM-SELF** message to the SSCP.

Local node detects the application's invalid use of ACKRDQ without the ECI application flag on a Data message



See Also

Concepts

Outbound Data

Inbound Data from LUA Applications

Inbound Data from LUA Applications

The local node performs certain checks on data supplied by a client application before sending it to the host and rejects it with a Status-Acknowledge(Nack-2) message if the checks fail. It does not return any acknowledgment to the application if the data passes the checks (although the host may do so later).

If the client application is providing a logical unit application (LUA) API, the design of the API may require that an LUA verb sending data inbound to the application does not complete until the local node has checked the data. Because of this, the local node will always respond to a client application that uses the LUA variant of the function management interface (FMI), after it has completed its send checking of the inbound message. This allows the client application to complete processing of the LUA verb and return control to the LUA application program.

If the inbound message passed the local node's send checks and will be sent to the host, the local node sends a Status-Acknowledge(ACKLUA) message to the client application to indicate this. The client application can then complete the LUA verb processing with an OK return code. Note that the **Status-Acknowledge(ACKLUA)** message does not imply that the data was successfully sent to the host or that the host received it. It may later be followed by a Status-Acknowledge(Nack-1) message indicating that the host rejected the data.

If the inbound message fails the local nodes send checking, a Status-Acknowledge(Nack-2) message will be returned as for non-LUA client applications. The client application can then report this to the LUA application program by a non-OK return code to the LUA verb that sent the message.

If the client application is providing an LUA API, it should therefore wait for either **Status-Acknowledge(ACKLUA)** or **Status-Acknowledge(Nack-2)** to determine whether to return an OK or error return code to the LUA send verb. If this dependence on the local node's send checks is not required, the client application can ignore the **Status-Acknowledge(ACKLUA)** message.

Note that there are certain race conditions in which the local node cannot complete its send checks before replying to the client application. In these cases, the local node returns a Status-Acknowledge(ACKLUA), but may subsequently send a Status-Acknowledge(Nack-2) if it detects an error during the remaining send checks. The client application may therefore receive a **Status-Acknowledge(ACKLUA)** followed by a **Status-Acknowledge(Nack-2)** for the same inbound message.

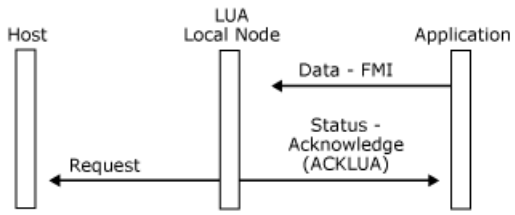
In the transmission header (TH) for the LUA variant of FMI, the expedited flow indicator (EFI), the destination-address field (DAF), and the origin-address field (OAF) are used. Other fields (including the sequence number field) are ignored. In the RH for the LUA variant of FMI, all fields except the queued-response indicator (QRI) and pacing indicator (PI) are used.

For inbound data, the first byte is TH[0] for the LUA variant of FMI.

The following three figures illustrate the **Status-Acknowledge(ACKLUA)** acknowledgment protocol for different messages that the application can send.

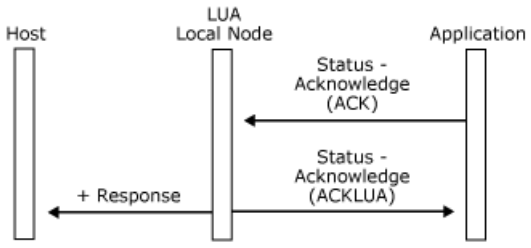
In the first figure, the application sends a **Data** message that passes the local node's send checks.

Application sends a Data message that passes the local node's send checks



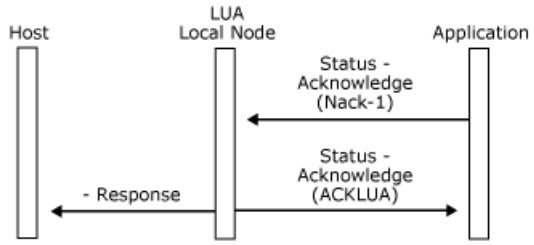
In the following figure, the application sends a **Status-Acknowledge(Ack)** message that passes the local node's send checks.

Application sends a Status-Acknowledge(Ack) message that passes the local node's send checks



In the following figure, the application sends a **Status-Acknowledge(Nack-1)** message that passes the local node's send checks.

Application sends a Status-Acknowledge(Nack-1) message that passes the local node's send checks



See Also

Concepts

Outbound Data

Inbound Data

Status Messages

The local node uses status messages to provide the application with information about the state of its sessions and to give the application control, in association with the host system services control point (SSCP) and primary logical unit (PLU), over the progress of the session. The status messages are designed to allow the application to use all the protocols allowed in the function management (FM) and Transmission Service profiles (TS profiles) supported by the Microsoft® Host Integration Server local node.

Most applications only need to use a subset of the available status messages. For example, a 3270 device emulator would not require the status messages used in quiesce protocols.

In This Section

Status-Acknowledge Message

Status-Acknowledge messages are the basic mechanism used to enforce inbound and outbound data acknowledgment protocols on all connections. For details about the use of **Status-Acknowledge** messages, see Outbound Data and Inbound Data.

For a 3270 emulation application, **Status-Acknowledge** messages sent from the application to the local node (acknowledging outbound data from the host) can also carry information about host response times. This allows the local node to maintain response time statistics for sending to the host when required. For details, see Response Time Monitor Data.

See Also

Reference

Status-Error Message
Status-Resource Message
Status-Session Message
Status-RTM Message

Other Resources

Status-Control Message

Status-Control Message

Status-Control messages provide access to session control and data flow control protocols on the primary logical unit (PLU) session using the PLU connection. They are not used on the other connections. **Status-Control** messages map directly to the equivalent SNA session control and data flow control request/response units (RUs).

All **Status-Control** messages that correspond to SNA requests on the normal flow with the exception of LUSTAT-sent Request Exception (RQE), and **Status-Control** messages corresponding to **CLEAR** and **STSN** request on the expedited flow, have the **ACKRQD** (acknowledgment required) field set. **Status-Control** messages that correspond to SNA requests on the expedited flow (with the exception of **CLEAR** and **STSN**) do not have the **ACKRQD** field set by the local node. However, the application can set **ACKRQD** when sending these **Status-Control** messages. The last figure in this topic summarizes which **Status-Control** requests always have **ACKRQD** set.

If a **Status-Control** request has **ACKRQD** set in the message header, the recipient must supply a **Status-Control** response (**Acknowledge, Negative-Acknowledge-1** or **Negative-Acknowledge-2**) before the sender sends further Data messages or further **Status-Control** requests on the flow. The sender can still send **Status-Control** responses, **Status-Acknowledge**, **Status-Error**, and **Status-Resource** messages on the flow. This applies to both normal and expedited flows and all request modes (including delayed-request mode). The message key received on the request must be returned on the response. (This is to allow multiple **RQE LUSTAT** messages to be outstanding.) The local node increments the message key on **Status-Control** requests and **DATAFMI** messages that it sends to the application on the PLU connection.

For the logical unit application (LUA) variant of the function management interface (FMI), the message key field is used in a different way, as follows:

- For inbound expedited flow requests, the local node sets the SNA sequence number to the value supplied by the application in the message key field. The application must ensure that this field is set to the correct sequence number.

- For inbound **Status-Control** responses, the local node sets the SNA sequence number to the value supplied by the application in the message key field. The application must ensure that this field is set to the sequence number of the request for which a response is being sent.

Except in the case of **Status-Control(LUSTAT)**, if a **Status-Control** request does not have **ACKRQD** set, the application should not reply, because a positive response has already been sent by the local node.

For example, if the application sends a **Status-Control(QC) Request** with **ACKRQD** set (corresponding to an SNA request on the normal flow), this blocks further data and **Status-Control** requests corresponding to the inbound normal flow until the **Status-Control(QC)** response is received. It does not block other messages on the normal flow, or messages on the expedited flow. For example, the application could still send **Status-Control(SIGNAL)**.

The receipt of the **Status-Control** response implies an acknowledgment to all outstanding messages (including **Data** messages) on the flow.

The use of **ACKRQD** on **Status-Control** messages effectively enforces definite-response and immediate request mode. This is appropriate for:

- **Status-Control** messages that correspond to the SNA requests **CLEAR** and **STSN** (because the expedited flow is **RQD**).

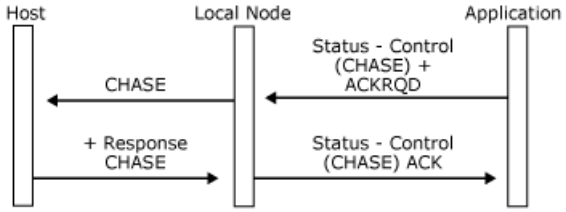
- **Status-Control** messages corresponding to all the **DFC** requests (which are **RQD**) except **LUSTAT** (which can be **RQE**).

The application can set **ACKRQD** on **Status-Control** requests that correspond to SNA requests on the expedited flow, even where **ACKRQD** is not required. For example, when an application is signaling for direction (for example, a 3270 emulator with a terminal operator repeatedly pressing the ATTN key), it can generate multiple **Status-Control(SIGNAL) Request** messages, which would adversely affect the performance of other users. The application can set **ACKRQD** on the first **Status-Control(SIGNAL) Request** and ignore events that would cause further **Status-Control(SIGNAL) Request** messages until the **Status-Control(SIGNAL) Response** is received from the local node.

The message flows in the following six figures show outbound and inbound **Status-Control** sequences with and without **ACKRQD** and the corresponding SNA RUs.

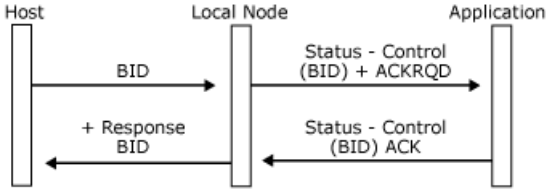
In the first figure, the application sends **Status-Control(CHASE)**.

Application sends Status-Control(CHASE)



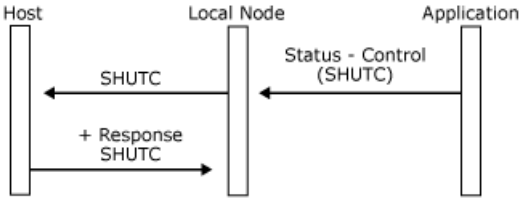
In the following figure, the host sends **BID** request.

Host sends BID request



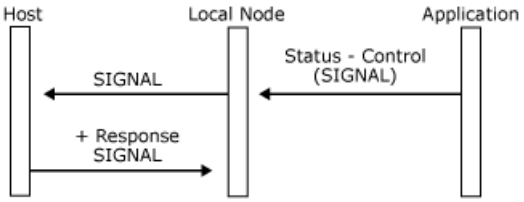
In the following figure, the application sends **Status-Control(SHUTC)**.

Application sends Status-Control(SHUTC)



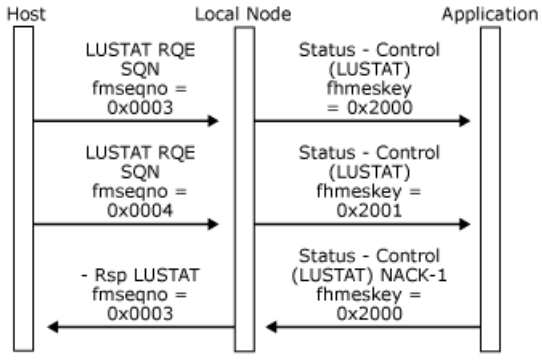
In the following figure, the host sends SNA **SIGNAL** request.

Host sends SNA SIGNAL request



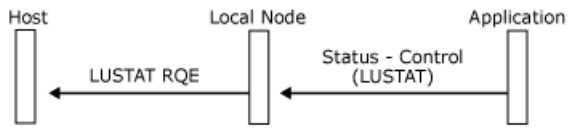
In the following figure, the host sends multiple **RQE LUSTAT** requests, and the application rejects the first one.

Application rejects the first RQE LUSTAT request



In the following figure, the application sends **Status-Control(LUSTAT) NOACKRQD**.

Application sends Status-Control(LUSTAT) NOACKRQD



The following table summarizes the **Status-Control** requests supported by the local node and SNA session control (SC) and data flow control (DFC) requests. For each **Status-Control** request, the table gives:

- The SNA category of the corresponding SNA request (SC or DFC).
- The flow used by the corresponding SNA request (normal or expedited).
- The TS or FM profiles on which the corresponding SNA request is supported.
- The directions for which it is valid (NODE <–> APPL).
- Whether it requires **ACKRQD**. Note that the application can set ACKRQD on a Status-Control request that does not require it.
- The hexadecimal code used in the control-type field of the Status-Control message. (For more information, see FMI Message Formats.)

Status-Control	SNA RQ flow	TS profile	FM profile	Direction node–appl	ACKRQD	Code
CLEAR	SC,Exp	2,3,4	–	–>	ACKRQD	CCLEAR (0x01)
SDT	SC,Exp	3,4	–	–>	–	CSDT (0x02)
RQR	SC,Exp	4	–	<–	–	CRQR (0x03)
STSN	SC,Exp	4	–	–>	ACKRQD	CSTSN (0x04)
CANCEL	DFC,Norm	–	3,4,7	<–>	ACKRQD	CCANCEL (0x10)
LUSTAT	DFC,Norm	–	3,4,7	<–>	–	CLUSTAT (0x11)
SIGNAL	DFC,Exp	–	3,4,7	<–>	–	CSIGNAL (0x12)
RSHUTD	DFC,Exp	–	3,4,7	<–	–	CRSHUTD (0x13)
BID	DFC,Norm	–	3,4	–>	ACKRQD	CBID (0x14)
CHASE	DFC,Norm	–	3,4	<–>	ACKRQD	CCHASE (0x15)
SHUTC	DFC,Exp	–	3,4	<–	–	CSHUTC (0x16)
SHUTD	DFC,Exp	–	3,4	–>	–	CSHUTD (0x17)
RTR	DFC,Norm	–	3,4	<–	ACKRQD	CRTR (0x18)
QC	DFC,Norm	–	4	<–>	ACKRQD	CQC (0x20)
QEC	DFC,Exp	–	4	<–>	–	CQEC (0x21)
RELQ	DFC,Exp	–	4	<–>	–	CRELQ (0x22)

The requests in the following table are used only with LUA. (For more information, see FMI Concepts.)

Status-Control	SNA RQ flow	TS profile	FM profile	Direction node–appl	ACKRQD	Code
CRV	SC,Exp	3,4	–	–>	ACKRQD	CCRV (0x05)
BIS	DFC,Norm	–	18	<–>	ACKRQD	CBIS (0x19)

SBI	DFC,Exp	–	18	<–>	ACKRQD	CSBI (0x1A)

The use of particular **Status-Control** messages is described in following topics of this section, in the context of PLU session protocols such as chaining, brackets, recovery, and so on.

For the formats of **Status-Control** messages, see Status-Control.

In This Section

- Status-Control (ACKLUA) Message

Status-Control (ACKLUA) Message

When a logical unit application (LUA) application sends a **Status-Control** message inbound to the local node, the LUA verb used to send the message cannot complete until the local node acknowledges the message. Because of this, the local node will always respond to the LUA application after it has completed its send checking of the inbound message. If the inbound message passes the local node's send checks, and the corresponding SNA message will be sent to the host, the local node sends a Status-Control(...) ACKLUA message to the application to indicate this. Note that the **ACKLUA** message does not imply that the SNA message was successfully sent to the host, or that the host received it.

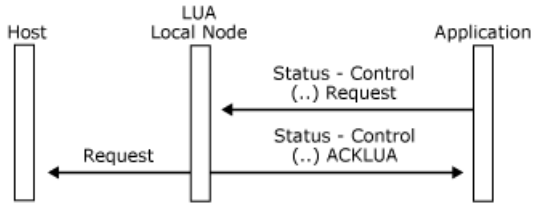
The format of the **Status-Control(...) ACKLUA** message is explained in Status-Control(...) ACKLUA. Note that the use of the message key field in **Status-Control(...) ACKLUA** is different from other **Status-Control** messages. It contains the sequence number from the transmission header (TH) of the **Status-Control** message sent by the LUA application, not the message key.

If the inbound message fails the local nodes send checking, a Status-Control(...) Negative-Acknowledge-2 message will be returned as for non-LUA applications. (This is then reported to the LUA application by a non-OK return code to the LUA verb that sent the message.)

The following three figures illustrate the **ACKLUA** acknowledgment protocol for different messages that the application can send.

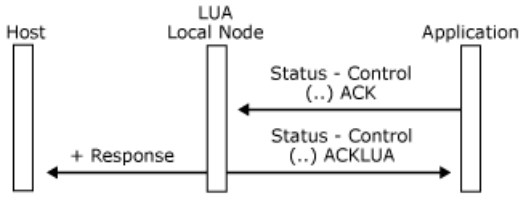
In the first figure, the application sends a **Status-Control(...) Request** message that passes the send checks of the local node.

Application sends a Status-Control() Request message



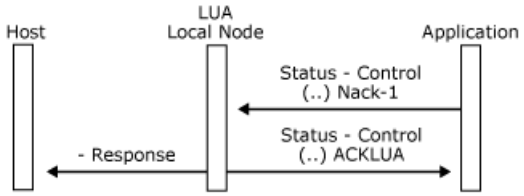
In the following figure, the application sends a **Status-Control(...) Acknowledge** message that passes the send checks of the local node.

Application sends a Status-Control(...) Acknowledge message



In the following figure, the application sends a **Status-Control(...) Negative-Acknowledge-1** message that passes the send checks of the local node.

Application sends a Status-Control() Negative Acknowledge-1 message



See Also

Reference

Status-Error Message

Status-Resource Message

Status-Session Message

Status-RTM Message

Other Resources

Status-Acknowledge Message

Status-Control Message

Status-Error Message

Status-Error messages flow from the local node to the application to report request reject and response header (RH) usage error conditions for:

- Errors in outbound expedited data flow control (DFC) requests.
- Errors in outbound session control (SC) requests.
- Errors in inbound responses.

The **Status-Error** message contains four bytes of error code information that contain the appropriate SNA sense codes for the detected error. For a list of error codes, see Error and Sense Codes.

See Also

Reference

Status-Resource Message
Status-Session Message
Status-RTM Message

Other Resources

Status-Acknowledge Message
Status-Control Message

Status-Resource Message

Status-Resource messages flow between an application and the local node to enable the local node to pace the primary logical unit (PLU) session of the application. They provide the local node with an indication of the buffer resources available at the application to receive outbound messages. With this information, the local node can determine when to send a pacing response. (For more information, see Pacing and Chunking.)

Note
Flow control is an option. Inbound pacing is handled by the local node, transparently to the application, and outbound pacing can be handled similarly. This is appropriate when only a limited number of messages flows from end-to-end of the SNA session between definite responses, such as with a 3270 screen.

See Also

Reference

Status-Error Message

Status-Session Message

Status-RTM Message

Other Resources

Status-Acknowledge Message

Status-Control Message

Status-Session Message

Status-Session messages always flow from the local node to the application and provide information about changes in the state of the session. There are separate **Status-Session** flows for each connection between the application and the local node.

The local node uses only one **Status-Session** message on the primary logical unit (PLU) connection. This is the **Status-Session(BETB)** message, used to report when the PLU session returns to the between-bracket state after the application or the PLU initiated a bracket. (For more information, see Brackets.)

The local node reports the activation and deactivation states of the system services control point (SSCP) session and PU-SSCP session using **Status-Session** messages. For example, it reports the receipt of an **ACTLU** request from the host SSCP using a **Status-Session (LU-Active)** message on the SSCP connection. (For more information, see SSCP Connection.)

By providing **Status-Session** messages rather than requiring the application to interpret the relevant information in the SNA request, the local node shields the application from decisions affecting conditional state transitions and from the necessity for a detailed understanding of SNA protocols.

A **Status-Session** message contains a status code, and for some status codes, an additional status code that qualifies the meaning of the primary status code. For example, the Link-Error status code, which occurs on the SSCP connection, is qualified by a status code that reports the link outage code supplied by the data link control layer of the local node. Applications such as 3270 device emulators use the qualifying status codes to display communications check codes (–+z_nnn) on the status line of the display.

The **Status-Session** codes are summarized in Status-Session Codes.

See Also

Reference

Status-Error Message
Status-Resource Message
Status-RTM Message

Other Resources

Status-Acknowledge Message
Status-Control Message

Status-RTM Message

The Status-RTM message is used by the local node to inform the application of the Response Time Monitor (RTM) parameters being used by the host. It flows from the local node to the application on the system services control point (SSCP) connection and is sent only for 3270 display logical units (LUs), or LUs in a pool of display LUs.

The **Status-RTM** message is sent at the following times:

- After the OK response to the Open(SSCP) Request message, to inform the application of the initial RTM parameters.
- When the RTM counters are reset, either due to a request from the host or when the local node sends unsolicited RTM data to the host.
- When any of the RTM parameters are changed by the host.

For more information about the use of the **Status-RTM** message, see RTM Parameters. For more information about how the application supplies RTM data to the local node, see Response Time Monitor Data.

See Also

Reference

Status-Error Message
Status-Resource Message
Status-Session Message

Other Resources

Status-Acknowledge Message
Status-Control Message

FMI Message Summary

This section lists each of the messages used at the function management interface (FMI) and gives a reference to a topic in the section where each message is described. The formats of the messages are given in FMI Message Formats.

For each message the direction of flow is indicated. IN means from the application to the local node, and OUT means from the local node to the application. The connection on which the message flows is also given.

Message	Direction	Connection	Reference
Open(SSCP) Request	IN	SSCP	Opening the SSCP Connection
Open(SSCP) Response	OUT	SSCP	Opening the SSCP Connection
Open(SSCP) Error Response	OUT	SSCP	Opening the SSCP Connection
Open(PLU) Request	OUT	PLU	Opening the PLU Connection
Open(PLU) OK Response	IN	PLU	Opening the PLU Connection
Open(PLU) Error Response	IN	PLU	Opening the PLU Connection
Open(PLU) OK Confirm	OUT	PLU	Opening the PLU Connection
Open(PLU) Error Confirm	OUT	PLU	Opening the PLU Connection
Close(SSCP) Request	IN	SSCP	Closing the SSCP Connection
Close(SSCP) OK Response	OUT	SSCP	Closing the SSCP Connection
Close(PLU) Request	IN/OUT	PLU	Closing the PLU Connection
Close(PLU) OK Response	IN/OUT	PLU	Closing the PLU Connection
Data-FMI	IN/OUT	SSCP/PLU	Data Flow
Status-Acknowledge(Ack)	IN/OUT	SSCP/PLU	Data Flow, Confirmation and Rejection of Data
Status-Acknowledge(Nack-1)	IN/OUT	SSCP/PLU	Data Flow, Confirmation and Rejection of Data
Status-Acknowledge(Nack-2)	OUT	SSCP/PLU	Inbound Data
Status-Control(CLEAR) Request	OUT	PLU	Recovery
Status-Control(CLEAR) Ack	IN	PLU	Recovery
Status-Control(CLEAR) Nack-1	IN	PLU	Recovery
Status-Control(SDT) Request	OUT	PLU	Status-Control Message
Status-Control(RQR) Request	IN	PLU	Recovery
Status-Control(RQR) Ack	OUT	PLU	Recovery
Status-Control(RQR) Nack-1	OUT	PLU	Recovery
Status-Control(RQR) Nack-2	OUT	PLU	Recovery
Status-Control(STSN) Request	OUT	PLU	Recovery
Status-Control(STSN) Ack	IN	PLU	Recovery
Status-Control(STSN) Nack-1	IN	PLU	Recovery
Status-Control(CANCEL) Request	IN/OUT	PLU	Outbound Chaining, Inbound Chaining

Status-Control(CANCEL) Ack	IN/OUT	PLU	Outbound Chaining, Inbound Chaining
Status-Control(CANCEL) Nack-1	IN/OUT	PLU	Outbound Chaining, Inbound Chaining
Status-Control(CANCEL) Nack-2	OUT	PLU	Inbound Chaining
Status-Control(LUSTAT) Request	IN/OUT	PLU	LUSTATs
Status-Control(LUSTAT) Ack	IN/OUT	PLU	LUSTATs
Status-Control(LUSTAT) Nack-1	IN/OUT	PLU	LUSTATs
Status-Control(LUSTAT) Nack-2	OUT	PLU	LUSTATs
Status-Control(SIGNAL) Request	IN/OUT	PLU	Direction
Status-Control(SIGNAL) Ack	OUT	PLU	Direction
Status-Control(SIGNAL) Nack-1	OUT	PLU	Direction
Status-Control(SIGNAL) Nack-2	OUT	PLU	Direction
Status-Control(RSHUTD) Request	IN	PLU	Application-Initiated Termination
Status-Control(RSHUTD) Ack	OUT	PLU	Application-Initiated Termination
Status-Control(RSHUTD) Nack-1	OUT	PLU	Application-Initiated Termination
Status-Control(RSHUTD) Nack-2	OUT	PLU	Application-Initiated Termination
Status-Control(BID) Request	OUT	PLU	Brackets
Status-Control(BID) Ack	IN	PLU	Brackets
Status-Control(BID) Nack-1	IN	PLU	Brackets
Status-Control(CHASE) Request	IN/OUT	PLU	Confirmation and Rejection of Data
Status-Control(CHASE) Ack	IN/OUT	PLU	Confirmation and Rejection of Data
Status-Control(CHASE) Nack-1	IN/OUT	PLU	Confirmation and Rejection of Data
Status-Control(CHASE) Nack-2	OUT	PLU	Confirmation and Rejection of Data
Status-Control(SHUTC) Request	IN	PLU	Shutdown and Quiesce
Status-Control(SHUTC) Ack	OUT	PLU	Shutdown and Quiesce
Status-Control(SHUTC) Nack-1	OUT	PLU	Shutdown and Quiesce
Status-Control(SHUTC) Nack-2	OUT	PLU	Shutdown and Quiesce
Status-Control(SHUTD) Request	OUT	PLU	Shutdown and Quiesce
Status-Control(RTR) Request	IN	PLU	Brackets
Status-Control(RTR) Ack	OUT	PLU	Brackets
Status-Control(RTR) Nack-1	OUT	PLU	Brackets
Status-Control(RTR) Nack-2	OUT	PLU	Brackets
Status-Control(QC) Request	IN/OUT	PLU	Shutdown and Quiesce
Status-Control(QC) Ack	IN/OUT	PLU	Shutdown and Quiesce

Status-Control(QC) Nack-1	IN/OUT	PLU	Shutdown and Quiesce
Status-Control(QC) Nack-2	OUT	PLU	Shutdown and Quiesce
Status-Control(QEC) Request	IN/OUT	PLU	Shutdown and Quiesce
Status-Control(QEC) Ack	OUT	PLU	Shutdown and Quiesce
Status-Control(QEC) Nack-1	OUT	PLU	Shutdown and Quiesce
Status-Control(QEC) Nack-2	OUT	PLU	Shutdown and Quiesce
Status-Control(RELQ) Request	IN/OUT	PLU	Shutdown and Quiesce
Status-Control(RELQ) Ack	OUT	PLU	Shutdown and Quiesce
Status-Control(RELQ) Nack-1	OUT	PLU	Shutdown and Quiesce
Status-Control(RELQ) Nack-2	OUT	PLU	Shutdown and Quiesce
Status-Error	OUT	SSCP/PLU	Status-Error Message
Status-Resource	IN	PLU	Pacing and Chunking
Status-Session	OUT	SSCP/PLU	Status-Session Message, Status-Session Codes
Status-RTM	OUT	SSCP	RTM Parameters

The following table lists the messages that are used for LUA only.

Message	Direction	Connection	Reference
Status-Acknowledge(ACKLUA)	OUT	SSCP/PLU	Inbound Data from LUA Applications
Status-Control(...) ACKLUA	OUT	PLU	Inbound Data from LUA Applications
Status-Control(CRV) Request	OUT	PLU	Status-Control Message
Status-Control(CRV) Ack	IN	PLU	Status-Control Message
Status-Control(CRV) Nack-1	IN	PLU	Status-Control Message
Status-Control(BIS) Request	IN/OUT	PLU	Status-Control Message
Status-Control(BIS) Ack	IN/OUT	PLU	Status-Control Message
Status-Control(BIS) Nack-1	IN/OUT	PLU	Status-Control Message
Status-Control(SBI) Request	IN/OUT	PLU	Status-Control Message
Status-Control(SBI) Ack	IN/OUT	PLU	Status-Control Message
Status-Control(SBI) Nack-1	IN/OUT	PLU	Status-Control Message

See Also
Reference
FMI Message Summary
Other Resources
FMI Concepts
SSCP Connection
PLU Connection
Data Flow
Status Messages

FMI Status, Error, and Sense Codes

This section lists the function management interface (FMI) status codes, error codes, and sense codes used on **Open** messages, **Status** messages, and **Data** messages with the system detected error indicator (SDI) set.

This section contains:

Status-Session Codes

For the status codes used on Status-Session messages, the following table lists:

- The value for the status code.
- The valid qualifying codes (if any) and their values.
- On which sessions the combinations of primary and qualifying status codes can occur.

For an overall description of the role of the **Status-Session** message, see Status-Session Message. The individual codes are discussed in SSCP Connection and PLU Connection.

Status code	Value	Qualifying code	Value	Usage
STNOSESS (no session)	0x01	STPUINAC STPUACT STPUREAC STLUINAC	0x10 0x03 0x04 0x11	SSCP SSCP SSCP SSCP
STLINERR (link error)	0x02	DLC ERROR	(See Note 1)	SSCP
STLUACT (LU active)	0x05	-	-	SSCP
STLUREAC (LU reactivated)	0x06	-	-	SSCP
STBETB (between brackets)	0x07	-	-	PLU

Note

The qualifying status code supplied on a **Status-Session** link error is the error code supplied by the data link control layer of the local node.

Note

Status-Session link error code 20 is generated by the node rather than by the link service. It indicates that the link service is not yet available, but is being activated. Ignore this error code during session activation. Otherwise, all **Status-Session** link errors (including 20) will cause the emulator to send a **Close(SSCP)**. When the emulator receives the Close(SSCP) Response, it will start again and send a new **Open(SSCP)**.

Note

The session status identifier is obsolete. It was used by the Microsoft® SNA Server OS/2 and Microsoft MS-DOS®-based 3270 emulators. Its value was obtained by adding 484 (decimal) to the qualifying status code shown in the preceding table.

Error and Sense Codes

This section describes the error and sense codes that are reported to the application in the following messages:

- Error Codes for Open(SSCP) Error Response
- Error Codes for Open(PLU) Error Confirm
- Error Codes for Nack-2 Messages
- Error Codes for Status-Error Messages
- Sense Codes for SDI Messages

Where the reported codes are SNA sense codes, a more complete description is given in Chapter 8 of the IBM document *Systems Network Architecture: Reference Summary* (GA27-3136). These SNA sense codes are also documented in Host Integration Server 2009 Help.

In addition, the local node delivers negative responses from the host as **Status-Acknowledge(Nack-1)** and **Status-Control(...) Negative-Acknowledge-1**, which can have any SNA sense code.

Application designers should note that error codes listed here that are specific to Host Integration Server always have an initial byte value of 0x00, and therefore can be easily distinguished from SNA sense codes, which have nonzero initial bytes.

The error codes are listed in topics for each type of message with an indication of the reason for the error.

This section contains:

- Error Codes for Open Messages
- Error Codes for Nack-2 Messages
- Error Codes for Status-Error Messages
- Sense Codes for SDI Messages

Error Codes for Open Messages

The possible error codes for **Open(SSCP) Error Response** and **Open(PLU) Error Confirm** are shown in the following topics.

This section contains:

- Error Codes for Open(SSCP) Error Response
- Error Codes for Open(PLU) Error Confirm

Error Codes for Open(SSCP) Error Response

The following table gives the values for error code 1 and error code 2 that can be returned on the **Open(SSCP) Error Response**.

Error code 1	Description	Error code 2	Description
0	No servers found.	CSRENOSR (0)	No servers found.
0x0053	Logical unit (LU) not verified.	CSRECBSH (3)	LU not verified.
0x0055	System services control point (SSCP) connection already open.	CSRECBSH (3)	Control block / resource shortage.
0x0057	No LU in group free.		
0x0812	No free session control block available.		
0x1001	A connection activation failed recently.		
0x1002	The connection is inactive.		
0x1008	The link service is active remotely. This error return code is not supported by this level of Host Integration Server.		
0x1009	The SNA server is active, but the connection on which the requested LU is defined is not active.		
0x100B	The connection is in the process of activating as the result of another **Open(SSCP)** or operator activation or recovery from an outage.		
0x1010	The connection is active, but an activate physical unit (ACTPU) has not yet been received.		
0x1011	The connection is active, but an activate logical unit (ACTLU) has not yet been received.		
0x0063	Unrecognized Open request.	CSRESERV (1)	Service not present.
0x0A0E	LU/LU group not found in configuration.		

Note

The error code 1 values 0x1001 to 0x1011 are returned when the Open(SSCP) Request specifies a nonforced Open. They do not indicate errors, but indicate that the LU-SSCP session is not active. The application can retry the **Open(SSCP) Request** specifying a forced Open, in which case the local node will attempt to activate the connection if possible.

Error Codes for Open(PLU) Error Confirm

The following table gives the values for error code 1 that can be returned on the Open(PLU) Error Confirm message. Error code 2 is zero, except when error code 1 is 0x0821. In this case it contains the byte offset in the **BIND** where the **BIND** failed to match the **BIND** check table.

Error code	Description
0x0051	Fewer than two buffer elements were present on **Open(PLU) Response**.
0x0052	System services control point (SSCP) connection is no longer active.
0x0821	**BIND** checking failed: error code 2 gives byte offset in **BIND** at which the error occurred.

Error Codes for Nack-2 Messages

The possible error codes delivered to the function management interface (FMI) application on Status-Acknowledge(Nack-2) and Status-Control(...) Negative-Acknowledge-2 messages are tabulated in the following table. A **Nack-2** is delivered to the application in response to data that is sent in error (or a Status-Control(...) Request that is in error). The data has not been sent to the host. The table indicates whether the error is critical, applying to the primary logical unit (PLU) connection only. If the error is critical, the critical failure indicator will be set in the message, and the application will receive a Close(PLU) Request as the next message.

All Nack-2 messages have the second word of information as 0x0000.

Error / Sense code	Critical YES/NO	Description
0x0040	YES	No buffer element on DATAFMI message.
0x0042	YES	DATAFMI message sent when no credit.
0x0043	YES	Invalid status-control for Transmission Service profile (TS profile).
0x0044	YES	Invalid status-control from application.
0x004A	YES	Half-duplex (HDX) contention and -QR,-BB,EB, or BKTFSM in pending-term-session.
0x0809	YES	Mode inconsistency.
0x1002	YES	Request/response unit (RU) length error.
0x1003	YES	Function not supported, invalid function management (FM) profile.
0x2002	NO	Chaining error.
0x2003	NO	Bracket error.
0x2004	NO	Direction error.
0x2005	YES	Data traffic reset.
0x2006	YES	Data traffic quiesced.
0x200D	YES	Response owed before sending request (half-duplex).
0x4003	YES	Begin bracket (BB) not allowed.
0x4004	YES	End bracket (EB) not allowed.
0x4006	YES	Exception response not allowed.
0x4007	YES	Definite response not allowed.
0x4009	YES	Change direction (CD) not allowed.
0x400A	YES	No-response not allowed.
0x400B	YES	Chaining not supported.

0x400C	YES	Brackets not supported.
0x400D	YES	CD not supported.
0x400F	YES	Incorrect use of FI.
0x4014	YES	Incorrect use of DR1, DR2, ER.
0x8005	NO	System services control point (SSCP) data sent when logical unit (LU) inactive.

Error Codes for Status-Error Messages

The possible error codes delivered to the function management interface (FMI) application on Status-Error messages are tabulated in the following table. A **Status-Error** message is delivered to the application in one of several cases, as shown in the following list:

- The local node detects an error in a response sent from the application (as a **Status-Acknowledge** or **Status-Control Ack/Nack-1** message).
- The local node detects an error in some data from the host that will not be delivered to the application as an system detected error indicator (SDI) message (such as an expedited flow request).
- The application sends an invalid **Status** message.

For inbound responses, the **Status-Error** codes have first byte 0x00. When the application is in error, the table indicates whether the error is critical, applying to the primary logical unit (PLU) connection only. If the error is critical, the application will receive a Close(PLU) Request as the next message.

The sense codes beginning with 0x40 will only be delivered if the corresponding receive check has been enabled in the connection information control block (CICB) on the Open(SSCP) Request from the application.

Where the sense code is marked with the * symbol, the second word of sense information carries the request code of the expedited flow request that was in error (for example 0x00C9 for SIGNAL).

Error / Sense code	Critical YES/NO	Description
0x0008	NO	Negative response already sent to this chain.
0x0040	YES	Invalid **Status** message from application.
0x0046	YES	Session failure due to correlation table shortage.
0x0050	YES	Invalid sequence number on Status-Ack.
0x0053	YES	Application may not send status control (**STSN**) negative acknowledge if it supports transaction numbers.
0x0056	YES	**Status-Ack** sent when previous RQD chains are outstanding. (For more information, see Outbound Data.)
0x0801	NO	Message received when pacing count is zero.
0x0805	NO	**BIND** from another PLU when already bound.
0x0809 *	NO	Mode inconsistency (**QEC** or **SHUTD**).
0x0815	NO	**BIND** from same PLU when already bound.
0x0821	NO	Incorrect **ACTLU** type (SSCP connection).
0x1003 *	NO	Wrong profile/network control request/invalid session control message.
0x2005	NO	Data traffic reset.
0x2007	NO	Data traffic not reset (**STSN** after **SDT**).
0x4009 *	NO	Change direction (CD) not allowed.
0x400B *	NO	Chaining not supported.
0x400C *	NO	Brackets not supported.

0x400F *	NO	Incorrect use of FI.
0x4011 *	NO	Incorrect use of request/response unit (RU) category.
0x4014 *	NO	Incorrect use of definite response 1 (DR1), definite response 2 (DR2), exception response (ER) .

Sense Codes for SDI Messages

When the local node detects an error in a normal flow request from the host, the message is converted into a **DATAFMI** message with the system detected error indicator (SDI) set to inform the application and to allow data to be processed serially. The application must convert the message to a Status-Acknowledge(Ack) to allow the local node to send the required negative response to the host. The possible error codes delivered to the function management interface (FMI) application on such SDI messages are tabulated in the following table.

The sense codes beginning with 0x40 will only be delivered if the corresponding receive check has been enabled in the connection information control block (CICB) on the Open(SSCP) Request from the application. If a receive check has been disabled, the message can still be converted to an SDI message. For example, a message with begin bracket (BB), -begin chain (BC) would fail as 2002 or 2003 if 4003 were disabled.

When the application uses a **Status-Control(LUSTAT) Request** to reject outbound data, the sense codes supplied by the application will be present on the SDI message generated by the local node. For more information, see LUSTATs.

Sense code	Description
0x0809	Mode inconsistency.
0x080B	Bracket race error.
0x081B	Contention race condition.
0x1003	Incorrect FM profile for request.
0x2001	Sequence number error.
0x2002	Chaining error.
0x2003	Bracket error.
0x2004	Direction error.
0x2006	Data traffic quiesced.
0x4003	BB not allowed.
0x4004	End bracket (EB) not allowed.
0x4006	Exception response not allowed.
0x4007	Definite response not allowed.
0x4009	Change direction (CD) not allowed.
0x400B	Chaining not supported.
0x400C	Brackets not supported.
0x400D	CD not supported.
0x400F	Incorrect use of FI.
0x4011	Incorrect use of RU category.
0x4014	Incorrect use of definite response 1 (DR1), definite response 2 (DR2), exception response (ER).

Configuration Information

To obtain information about the Microsoft® Host Integration Server 3270 configuration, the application uses the calls listed in the following table.

Function	Description
sepdcrec	Returns a data structure that contains the 3270 user record for this user and the diagnostics record from the running configuration file.
sepdgetinfo	Returns general information about the version of Host Integration Server currently running, such as the release level, the network operating system, and the directory of the running configuration file.

If the return code from **sepdcrec** indicates that no 3270 user record was found for this user, the emulation program should terminate and not allow the user to use 3270 emulation. The Host Integration Server error message COM0438 is provided to log this error.

In This Section

- 3270 User Record Format
- Diagnostics Record Format
- Creating NetView User Alerts

3270 User Record Format

Two structures make up the 2370 user record:

Tecwrksd, a logical unit (LU)/session information record, which includes details of a 3270 LU.

Tecwrkus, a 3270 user record, which includes a number of Tecwrksd LU/session information records.

Note that the user record is not a fixed length, because the number of LU/session information records in the remap list is variable. The structures described in the following sections are provided simply as a template to allow you to map to the correct offset in the record.

In This Section

tecwrksd

Tecwrksd is a logical unit (LU)/session information record, which includes details of a 3270 LU.

Syntax

```
typedef struct tecwrksd {
    UCHAR   cwshost[9];
    USHORT  cwsestyp;
    USHORT  cwsmodov;
    USHORT  cwspad;
} TECWRKSD;
```

Members

cwshost[9]

LU/pool name accessed.

cwsestyp

Session type (M2, M3, M4, M5, printer).

cwsmodov

Whether the user has override permission.

cwspad

Two bytes of padding.

Remarks

For Windows 2000

The following "Members" list explains the meaning of each field in the structures and indicates how the application should use each field. For more information about Host Integration Server 3270 configuration, see Configuration Information.

Members

cwshost

The name (up to eight characters) of the LU or LU pool that this session is configured to use. The application specifies this name on the Open(SSCP) Request.

cwsestyp

The LU type (display or printer) of the LU used by this session and (if it is a display LU or a pool of display LUs) the screen model. The possible values are:

- CERTMOD2 (0) Model 2 display (24 by 80)
- CERTMOD3 (1) Model 3 display (32 by 80)
- CERTMOD4 (2) Model 4 display (43 by 80)
- CERTMOD5 (3) Model 5 display (27 by 132)
- CERTPRNT (4) Host printer

The application should use this value to distinguish between display and printer sessions and to set the appropriate screen model for display sessions.

cwsmodov

TRUE if the user has permission to override the screen model for display sessions—that is, to change the session to use a different screen model from the one configured. If this value is **FALSE**, the user should not be permitted to change the screen model. This field is not used for printer sessions and should not be checked.

tecwrkus

Tecwrkus is a 3270 user record, which includes a number of Tecwrksd LU/session information records.

Syntax

```
typedef struct tecwrkus {
    USHORT   cwlen;
    USHORT   cwtype;
    UCHAR    cwname[21];
    UCHAR    cwremark[26];
    UCHAR    cwstylef[9];
    USHORT   cwvewrtm;
    USHORT   cwalert;
    USHORT   cwchghan;
    USHORT   cwmaxses;
    USHORT   cwnumrec;
    TECWRKSD cwsesdat[10];
    USHORT   cwmodisf;
    UCHAR    cwstatus;
    UCHAR    cwpad;
    USHORT   cwnumrmp;
    TECWRKSD cwremap[1];
} TECWRKUS;
```

Members

cwlen

Length of record.

cwtype

Type of record.

cwname[21]

User name.

cwremark[26]

Comment field.

cwstylef[9]

Initial style file name.

cwvewrtm

Whether user can view Response Time Monitor (RTM) information.

cwalert

Whether user has ALERT permission.

cwchghan

Whether user can change LU/pool name accessed.

cwmaxses

Maximum number of active sessions (1–10).

cwnumrec

Number of sessions for user.

cwsesdat[10]

Session information records.

cwmodisf

Permission to modify initial style.

cwstatus

Status byte: user or group.

cwpad

One byte of padding.

cwnumrmp

Number of LUs/pools in remap list.

cwremap[1]

LU/pool remap list.

Remarks

For Windows 2000

Microsoft® Host Integration Server permits configuration of more than 10 sessions per user when used with clients running Microsoft Windows® 2000. The first 10 sessions are placed in the **cwsesdat** array with **cwnumrec** set to 10 and the remainder are placed in the location of the remap list. The **cwnumrmp** member indicates the number of Tecwrksd structures in the remap list. Note that this permits **cwmaxses** to be greater than **cwnumrec**.

The following "Members" list explains the meaning of each field in the structures and indicates how the application should use each field. For more information about Host Integration Server 3270 configuration, see Configuration Information.

Members

cwlen

The length of the 3270 user record (this is variable because it contains a variable number of LU/session records in the remap list). The application should use this value to locate the start of the next 3270 user record when searching for the correct record.

cwtype

Identifies this as a 3270 user record.

cwname

The Local Area Network (LAN) Manager user name, or other identifying name, of the 3270 user (up to 20 characters). The application uses this to search for the correct 3270 user record.

cwremark

An optional comment field (up to 25 characters), used in the configuration program to give more information about the user (for example, the user's full name).

cwstylef

The name (up to eight characters) of the default style file used by this user (a file containing the user's 3270 customization settings, used by the Host Integration Server 3270 emulation programs). This field can be used to identify the equivalent file for your 3270 emulator, if appropriate.

If this field is blank, no style file is used and the 3270 emulator should revert to its default settings (unless overridden by a style file specified by the user).

cwvewrtm

TRUE if this user is permitted to view a display of RTM statistics for his or her 3270 sessions. If this field is **FALSE**, the application should not display RTM statistics and should not display a last transaction time indicator (LTTI) on the status line of display sessions. For more information about the use of Response Time Monitor (RTM), see Diagnostics Record Format.

cwalert

TRUE if the user is permitted to send NetView user alerts. If this field is **FALSE**, the user should not be permitted to send alerts. For more information about the use of alerts, see Diagnostics Record Format.

cwchghan

TRUE if the user is permitted to remap a 3270 session to use a different LU (in which case it can be changed to use any LU in the remap list—see **cwremap**). If this field is **FALSE**, the application should not allow the user to remap sessions.

cwmaxses

The maximum number of active sessions permitted to this user. If the number of sessions configured (see **cwnumrec**) is greater than this, the user must not be allowed to activate more sessions at a time than this field specifies.

cwnumrec

The total number of sessions configured for this user. The user record always contains 10 LU/session records (see **cwsesdat**), but only this number of the records will be used—the remainder will be filled with zeros.

cwsesdat

Ten LU/session records. Some of these records can be filled with zeros, indicating that they are unused (**cwnumrec** gives the number of sessions that are used). The application should list, and allow the user to use, only the sessions that have valid session records here.

cwmodisf

TRUE if the user is permitted to modify the initial 3270 customization. If this field is **FALSE**, the application should use the customization defined by **cwstylef** (if specified); the user should not be allowed to make changes to this style, or to override it by loading a different style file.

cwstatus

Indicates whether the user name in this record is a LAN Manager user name or group name. The least significant bit of this byte is **CERTGRUP (1)** for a group, and **zero** for a user. Other bits are not used.

cwpad

Pad byte—not used by the application.

cwnumrmp

The number of LU/session records in the remap list (see **cwremap**).

cwremap

The list of LU/session records, which indicates the LUs to which the user can remap sessions (if any). If the user is not permitted to remap sessions (see **cwchghan**), this list is not used and should not be checked by the application.

Diagnostics Record Format

Two structures make up the Diagnostics record:

The Alert information record tedalert, which includes details of a 3270 NetView user alert, and tediagns, the diagnostics record, which includes a number of tedalert information records.

In This Section

- tedalert
- tediagns

Tedalert

The alert information record tedalert includes details of a 3270 NetView user alert.

Syntax

```
typedef struct tedalert {
    UCHAR    dalrtnam[53];
    UCHAR    daparam1[33];
    UCHAR    daparam2[33];
    UCHAR    daparam3[33];
} TEDALERT;
```

Remarks

The following "Members" list explains the meaning of each field in the structures that is relevant to the application and indicates how the application should use each field. Fields that are not included in the list are used by other Microsoft® Host Integration Server components and need not concern the application; in particular, the network management connection name and the times at which RTM data is sent to the host are handled by the local node on behalf of the application.

Note that the application should determine whether the user is permitted to send NetView user alerts and/or view RTM data (see 3270 User Record Format). It should not display the appropriate information, as described below, if the user does not have permission to use that information. The host can also override whether the application is permitted to send and/or to display RTM data (for more information, see RTM Parameters).

For more information about how the application uses the RTM parameters, see RTM Parameters, Response Time Monitor Data, and Status-RTM.

Members

dalrtnam[53]

The description (up to 52 characters) of the alert corresponding to a particular alert number. The application should display this information to help the user determine which alert to send.

daparam1[33]

The descriptions (each up to 32 characters) of up to three parameters.

daparam2[33]

Required for the alert; depending on the specific alert.

daparam3[33]

One or more of these descriptions can be blank, indicating that the parameter is not used. For each of these descriptions that is not blank, the application should display this string to prompt the user for the appropriate parameter.

tediagns

The diagnostics record tediagns includes a number of tedalert information records.

Syntax

```
typedef struct tediagns {
    USHORT   dilen;
    USHORT   ditype;
    UCHAR    dinetmgt[9];
    USHORT   disrtmco;
    USHORT   disrtmub;
    USHORT   diwruldr;
    USHORT   dirtmth1;
    USHORT   dirtmth2;
    USHORT   dirtmth3;
    USHORT   dirtmth4;
    TEDALERT dialerts[20];
    UCHAR    diaudit[128];
    UCHAR    dierror[128];
    USHORT   diaudlev;
    UCHAR    dipad[16];
} TEDIAGNS;
```

Members

dilen

Length of record.

ditype

Type of record.

dinetmgt[9]

Network management connection name.

disrtmco

Send Response Time Monitor (RTM) data at counter overflow.

disrtmub

Send RTM data at UNBIND.

diwruldr

The definition by which response times are to be measured. The application should measure the response time from the time the user presses ENTER or an AID key to send data to the host, until one of the following events as defined by this field:

CERTWRIT (0)

The first host data reaches the 3270 display.

CERTUNLK (1)

The host unlocks the user's keyboard.

CERTDIRE (2)

The host gives the application direction so that the user can send further data.

Note that the host can override these definitions; for more information, see RTM Parameters.

dirtmth1

The thresholds that define the bands into which response times are to be classified. Note that the host can override these definitions; for more information, see RTM Parameters.

dirtmth2

RTM threshold #2.

dirtmth3

RTM threshold #3.

dirtmth4

RTM threshold #4.

dialerts[20]

Up to 20 alert records that define the alerts that Host Integration Server users can send to a host. There are always 20 records, but some of these can be blank, indicating that they are not used. The application should display the descriptions of any nonblank alerts together with the alert number (from 1 to 20) defined by the position of the alert record in this array.

diaudit[128]

Audit log file name.

dierror[128]

Error log file name.

diaudlev

Default audit level.

dipad[16]

16 bytes of padding.

Creating NetView User Alerts

You can create NetView user alerts for users to send. Users identify the alerts by number. Each number corresponds to a specific collection of information or requests that the user wants to send through NetView to a host operator.

Microsoft® Host Integration Server leaves blank fields for the user alert information in the structure that is returned from sepdcrec. To create specific user alerts, create appropriate data structures and call the TRANSFER_MS_DATA common service verb to send the user alert to NetView.

Compiling and Linking 3270 Client Applications

This section describes the 3270 client samples included with Host Integration Server 2009. The samples are located in the \SDK\Samples\SNA folder on the Host Integration Server CD.

This section lists and explains the header files and libraries needed to develop 3270 client applications for use with Host Integration Server client applications. It also provides information about compiling and linking the 3270 client applications.

In This Section

Building the 3270 Client Samples

When installed from the Microsoft Host Integration Server 2009 CD, all the 3270 client samples are built in a similar way, as described in this section. First, set the environment variables listed in the following table.

Variable	Description
ISVLIBS	The directory containing the Host Integration Server LIB files for Microsoft Windows 2000.
ISVINCS	The directory containing the Host Integration Server header files.
SAMPLEROOT	The root directory where the sample code provided as part of the SDK has been installed on a local hard disk.

For example, after copying the contents of the SDK folder on the Host Integration Server 2009 CD to C:\SNASDK, use the following lines to set the variables:

```
ISVLIBS=C:\SNASDK\Lib
ISVINCS=C:\ SNASDK\Include
SAMPLEROOT=C:\SNASDK\Samples\SNA
```

Next, run NMAKE on the .mak file in each subdirectory below this root directory containing the actual sample source code. For example, for APING and APINGD, change to the Samples\aping directory and type the following:

nmake -f makeping.mak

Client Interface Files for 3270 Applications

The files listed in the following table are required to build 3270 client applications for use with Microsoft® Host Integration Server.

File	Description
FMI.H	Main header file containing the definitions of buffer and message formats, function prototypes for the DL-BASE/DMOD interface calls, and constant definitions.
TRACE.H	Definitions of the logging and tracing macros.
FMISTR32.LIB	Function Management Interface (FMI) string library for use with Microsoft Windows® 2000.
SNACLI.LIB	Main interface library for developing 3270 client applications on Windows 2000.

3270 Include Files

To compile the application, the header files FMI.H and TRACE.H are required. In addition, one of the standard operating system header files may be required. To include the required files, use the following lines in your application:

```
#include <fmi.h>
#include <trace.h>
```

Compiler Options for 3270 Applications

When compiling the 3270 client application, the compiler options listed in the following table are required.

Option	Explanation
/c	Compile only, without linking. Linking is normally done as a separate phase to include the required Microsoft® Host Integration Server libraries.
/D NOTRC	The NOTRC macro specifies that internal tracing should not be compiled into the application. The **/D NOTRC** option should be used for building a final system (internal tracing should not be included because it will degrade performance and require more memory and resources). For a development system, you may want to compile with internal tracing; if so, remove the **/D NOTRC** option.
/D WIN32_SUPPORT /D MSWIN_SUPPORT, /D OS2_SUPPORT, /D DOS_SUPPORT	These macros are used in the header files FMI.H and TRACE.H supplied with SNA services to support variants of the client interface for the different operating systems supported. One of these options must be defined, depending on the operating system for which the application is intended.

/Gzs	c:	Use **stdcall** calling conventions on i386/i486 and Pentium class processors.
	S:	Remove stack check calls.

The compiler flags listed in the following table are required, but any of the valid options for each flag may be used, as appropriate to your application.

Flag	Description
/A	Compiler model (Does not apply to Microsoft Windows® 2000)
/O	Optimization
/W	Warning level

Linking 3270 Client Applications

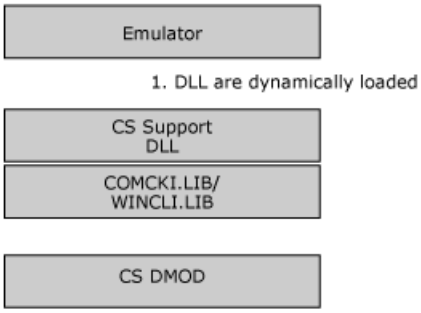
This topic describes how to link 3270 client applications using different platforms.

The SNACLI.LIB library must be linked with the application.

The DMOD is implemented as a DLL. SNACLI.LIB contains import definitions for the APIs in the DLL, and some global variables required for the logging and tracing macros.

It is possible to create a DLL that is dynamically loaded when the user starts a session for an LU. In this case, to make the log and trace macros available, the application structure needs to be as shown in the following figure.

Application Structure



Support for 3270 Single Sign-On

This section describes the support for Single Sign-On for 3270 display sessions that is available in Microsoft Host Integration Server 2009.

Over 3270 LUs, a Single Sign-On feature is supported to automate the overall logon process. When configured for this feature, Host Integration Server automatically replaces special keywords in the data stream with the actual host user name and password at appropriate points in the session.

This section contains:

Prerequisites for 3270 Single Sign-On

In preparation for using 3270 Single Sign-On, the system administrator must define a host security domain containing host connections. This host security domain must be initially created or modified to enable the Single Sign-On feature. The system administrator must enable a user's Microsoft® Windows Server™ 2003 or Microsoft Windows® 2000 account in the host security domain and either the administrator or the user must establish a mapped host account for the Microsoft® Windows Server™ 2003 or Windows 2000 domain user name.

The user must be logged on to a Microsoft® Windows Server™ 2003 or Windows 2000 domain with a user name and password. Note that this Single Sign-On feature is only supported over 3270 LUs.

Registry Settings Used for 3270 Single Sign-On

The 3270 Single Sign-On feature depends on Microsoft® Host Integration Server scanning 3270 logical units (LUs) used in the logon process for special keywords that are defined in the registry on the computer running Host Integration Server. The values for these special keywords can be defined by the system administrator on the computer running Host Integration Server.

The registry settings used by the 3270 Single Sign-On process are located under the **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services** registry node. The following entries are installed under the **SNASERVR\PARAMETERS** subkey:

3270SSOPadByte
This entry should be set to an ASCIIZ string to use as the character for padding replacement text in the user name or password if these strings are shorter than the length of the special tag strings defined below. The default value for this pad character is the ASCII space character.

3270SSOPostReplaceCount
This entry should be set to a DWORD that represents the number of message chains of RUs to scan after replacement of text for user name or password. The default value for this number is 10.

3270SSOPrefix
This entry should be set to an ASCIIZ string to use as the special prefix tag string in combination with the user name and password tags. The default value of this string is MS$.

3270SSOPwdTag
This entry should be set to an ASCIIZ string to use as the special tag string in combination with the **3270SSOPrefix** tag in defining the special host password string that will be replaced. The default value of this string is SAMEP, so the default host password string that is scanned for and replaced is MS$SAMEP. Note that length of the password string that is scanned for (MS$SAMEP, for example) determines the maximum length of the password string that can sent to the host using Single Sign-On. This limit occurs because the password substitution cannot change the length of the data message. Note that the value of this string must be different from the value of the **3270SSOUserTag** entry for Single Sign-On to function properly.

3270SSOReplaceCount
A DWORD value that affects the time-out value for password substitution. User IDs and passwords will be substituted in each chain on the LU-SSCP and PLU-SLU sessions until the timer expires. By default the timer will be set to 30 seconds, but this behavior can reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node.

If the **3270SSOReplaceCount** registry entry is defined and the **3270SSOReplaceTimer** registry entry is not defined, the node counts this number of RUs (on PLU-SLU session only) before time-out occurs. If both the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries are defined, the value for **3270SSOReplaceCount** will be used to determine when a time-out will occur. By default, this key is not defined and the node defaults to a time-out of 30 seconds.

3270SSOReplaceTimer
A DWORD value that affects the time-out value for password substitution. User IDs and passwords will be substituted in each chain on the LU-SSCP and PLU-SLU sessions until the timer expires. By default the timer will be set to 30 seconds, but this behavior can reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node.

If the **3270SSOReplaceTimer** registry entry is defined and **3270SSOReplaceCount** is not defined, the node uses this value in seconds before time-out occurs. If both the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries are defined, the value for **3270SSOReplaceCount** will be used to determine when a time-out will occur. By default, this key is not defined and the node defaults to a time-out of 30 seconds.

3270SSOUserTag
This entry should be set to an ASCIIZ string to use as the special tag string in combination with the **3270SSOPrefix** tag in defining the special user name string that will be replaced. The default value of this string is SAMEU, so the default user name string that is scanned for and replaced is MS$SAMEU. Note that length of the user name string that is scanned for (MS$SAMEU, for example) determines the maximum length of the username string that can sent to the host using Single Sign-On. This limit occurs because the user name substitution cannot change the length of the data message. Note that the value of this string must be different from the value of the **3270SSOPwdTag** entry for Single Sign-On to function properly.

3270 User Name and Password Replacement

The SNA node on the host monitors the inbound session for a replacement sequence consisting of the **3270SSOPrefix** string immediately followed by one of the strings **3270SSOUserTag** or **3270SSOPwdTag**. Thus, the default user name string that would be scanned for and replaced is MS$SAMEU. When this string is found in the inbound session data, the node looks up the corresponding information (host user name in the current host security domain) and overwrites MS$SAMEU with this information. The same process occurs for the password string that would be scanned for and replaced, which defaults to MS$SAMEP.

Note that this operation cannot change the length of the data message. If the actual user name or password that is retrieved from the current host security domain is shorter than the replacement sequence, it is padded out with the first character of the **3270SSOPadByte** string used as a padding character. If the actual host user name or password string is longer than the string that is scanned for, these strings are truncated to the length of the scanned string so that the data message length is not affected.

Note that since the user name and password can be sent in any order, the registry string values for the **3270SSOUserTag** and **3270SSOPwdTag** entries must be different for Single Sign-On to function properly.

The SNA node monitors the SSCP-LU session for these special tag strings at all times and replaces all occurrences of these strings with corresponding looked-up data. On the LU-LU session, the node starts monitoring at start of session (BIND). The node stops monitoring when it has received **3270SSOPostReplaceCount** chains of request/response units (RUs) without seeing a substitution tag. The node will not restart monitoring until it receives an UNBIND–BIND sequence for that session.

Note that the node considers the sequence:

```
BIND, data, UNBIND(BIND FORTHCOMING), BIND
```

As a continuation of the same LU-LU session and does not restart monitoring on receipt of the second BIND. This sequence is often used by host session managers handing off a session to an application system, and is considered a single terminal session.

User IDs and passwords will be substituted in each chain on the system services control LU-SSCP and PLU-SLU sessions until the node has received **3270SSOPostReplaceCount** chains of RUs without seeing a substitution tag or a timer expires. By default the timer is set to 30 seconds, but this behavior can reconfigured in the registry using the **3270SSOReplaceCount** and **3270SSOReplaceTimer** registry entries. The timer is started when the OPEN SSCP is received by the node. After the timer expires, the node will stop scanning messages for the 3270 replacements strings for the user ID and password. If the replacement strings arrive after the timer expires, the replacement strings will be sent to the host unmodified causing the Single Sign-On to fail. The application will not receive any notification that the timer has expired. The only indication of a problem will likely be that the Single Sign-On to the host session has failed.

Note that all strings are specified in the registry in ASCII, but the node translates them to Extended Binary Coded Decimal Interchange Code (EBCDIC) through AE character mapping before scanning for a match.

SNA Internationalization Programmer's Guide

This section describes the features available in Host Integration Server 2009 for supporting international languages and different national language character sets.

The SNANLS API uses the language support features provided with Microsoft Windows 2000 and Microsoft Windows XP. SNANLS supports European languages that use single-byte encoding, as well as East Asian languages that use double-byte or Unicode encoding.

For API references and other technical information about SNA Internationalization, see SNA Internationalization Programmer's Reference.

In This Section

- SNA National Language Support Programmer's Guide

SNA National Language Support Programmer's Guide

The SNA National Language Support (SNANLS) API standardizes the way in which national languages and locales are supported. SNANLS handles string conversion necessary for supporting a wide range of host and code pages. Components such as the Host Print service and Shared Folders service use SNANLS API to convert strings from EBCDIC to ANSI and from ANSI to EBCDIC.

The SNANLS API is the standard means to convert strings in Host Integration Server 2009. SNANLS presents a single interface to applications that need strings converted from one code page to another. These conversions may be EBCDIC-to-ANSI, ANSI-to-EBCDIC, EBCDIC-to-OEM code pages, OEM-to-EBCDIC, EBCDIC-to-ISO code pages, and ISO-to-EBCDIC. Additionally, SNANLS supports the broadest possible range of host and PC code page conversions.

SNANLS provides a uniform interface for programmers, hiding the details and difficulties of string conversion. SNANLS supports both SBCS and DBCS conversions. Two other lower-level APIs handle the actual string conversion. For SBCS conversions, SNANLS uses the system-provided Win32 NLS API that is resident on Microsoft Server 2003, Windows XP, and Windows 2000.

For DBCS conversions, SNANLS uses the **TrnsDT** API. The **TrnsDT** API is installed with Host Integration Server.

SNANLS is supported on Windows Server 2003, Windows XP, and Windows 2000.

In This Section

National Language Support in Windows Server 2003, Windows XP, and Windows 2000

National Language Support (NLS) provides a standardized method of supporting multiple international locales, code pages, input methods, sort orders, and number/currency/time/date formats.

The Win32 NLS API provides developers with a way to access system-provided Unicode-to-ANSI and ANSI-to-Unicode conversion services. Microsoft Windows Server 2003, Windows XP, and Windows 2000 supply EBCDIC-to-Unicode and Unicode-to-EBCDIC translation tables for all of the popular host code pages.

The SNANLS API leverages the existing work done to support the NLS API on Windows Server 2003, Windows XP, and Windows 2000. Host Integration Server 2009 takes advantage of these EBCDIC-to-Unicode-to-ANSI and ANSI-to-Unicode-to-EBCDIC code page conversion services.

Currently, the Win32 NLS API only supports SBCS EBCDIC code pages. However, future versions of the NLS API will support DBCS EBCDIC. SNANLS currently uses TrnsDT for DBCS conversions.

SNA Print Server Data Filter Programmer's Guide

The Host Print service of Microsoft® Host Integration Server 2009 provides server-based 3270 and 5250 printer emulation, enabling host applications to print to Local Area Network (LAN) printers supported by Microsoft Windows Server™ and Novell NetWare. This section introduces the SNA Print Server Data Filter API (sometimes referred to as the Print Exit API). This API can be used to extend the capabilities of the Host Print service in Host Integration Server 2009.

The user can provide a print data filter DLL that will be called by Host Print service when a print job is initiated, when data is sent to the printer, and when the print job is completed. This print data filter DLL can:

- Send data to the printer when a job starts (print a banner page, for example).
- Perform special processing on the data to be printed.
- Send data to the printer upon print job completion (print a trailer page, for example).

SNADIS Programmer's Guide

This section enables original equipment manufacturers (OEMs) and adapter vendors who are developing their own SNALink software to work with Host Integration Server 2009 SNA Services.

SNALink software must be written as a Microsoft Windows 2000 device driver. For information about writing Windows 2000 device drivers, see the Windows 2000 Device Driver Kit (DDK).

This section provides the following information:

- Internal concepts of Host Integration Server 2009 that are required to integrate new communications adapters into the server environment.
- Definitions of the interfaces used by Host Integration Server 2009 to communicate with SNALinks.
- Information about using the configuration and diagnostics features included in Host Integration Server 2009.
- Instructions for compiling and linking the SNALink support software.

The network operating systems currently supported by Host Integration Server 2009 include Microsoft LAN Manager (as implemented in Microsoft Windows 2000) and TCP/IP. Future versions of Host Integration Server may support other network operating systems. Because of this, it is recommended that you develop link support that is independent of the network operating system.

For API references and other technical materials for programming SNALink applications, see the SNADIS Drivers Programmer's Reference section of the SDK.

This section contains:

SNALink Concepts in Host Integration Server

This section describes key concepts used in the SNALink feature of Host Integration Server 2009. Because the purpose of this section is to enable original equipment manufacturers (OEMs) and adapter vendors to develop link support software to integrate their hardware adapters into a Host Integration Server 2009 system, only the relevant parts of the Host Integration Server 2009 architecture are described.

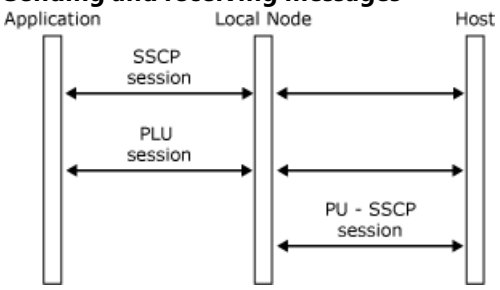
In This Section

Overview of SNALink

A Host Integration Server 2009 SNALink must implement an SNA-compatible data transport mechanism capable of connecting the local type 2.1 node to remote host (PU4/5) or peer (PU2.1) systems.

The local node provides the SNA layers of path control, transmission control, data flow control, and logical unit (LU) services. The following figure shows an example of a Host Integration Server 2009 system.

Sending and receiving messages



The local node uses the data link control (DLC) interface to communicate with a SNALink. This interface is defined in The Data Link Control Interface. The SNALink and the DLC driver are responsible for transferring data between the path control layer of the node and the DLC adapter.

The routing of messages that flow between Host Integration Server components is handled by the SnaBase and Dynamic Access Module (DMOD) components. For details about how to send and receive messages, see The SNALink Interface.

SNALink Configuration and Management

The configuration information for a Host Integration Server 2009 system is stored in two forms:

- A centralized configuration file containing details of logical units (LUs), physical units (PUs), and connections.
- Entries in the Microsoft Windows Server 2003 or Windows 2000 registry containing configuration information for the SNALinks supported on that computer. This information contains parameters required by Host Integration Server 2009, and any other parameters that independent hardware vendor (IHV) code may require.

A Host Integration Server SNALink is defined when a Host Integration Server system is installed. A SNALink can support only one physical connection from the server. If a single adapter is capable of supporting multiple physical connections, Host Integration Server requires multiple SNALinks to be configured.

To reconfigure a server's SNALink support (for example, after installing a new adapter), the administrator uses either the Windows Network Control Panel applet or the Host Integration Server setup program. For further information about how this operates, see Setup Information.

All other configuration of a Host Integration Server system is performed using SNA Manager. As part of the configuration process, logical connections to remote PUs are associated with one or more SNALinks.

All configured SNALinks are automatically started when the Host Integration Server system is started. At this stage, the SNALink performs any initialization required, and then waits for instructions from local nodes.

When a connection is activated, either from SNA Manager or automatically (for example, in response to a 3270 user's request for a session with a remote host), the SNALink receives an Open(LINK) message from the local node. The SNALink should then perform whatever action is required to initiate that connection. This can involve dialing a telephone number for a switched Synchronous Data Link Control (SDLC) connection or bringing up level 2 on an X.25 link and sending a CALL packet.

If the IHV wants the same physical adapter to be available for use by multiple SNALinks (for example, a dumb SDLC card can be used to communicate using SDLC or X.25 protocols), the SNALink should not attempt to access the hardware until it has received an **Open(LINK)** message from the local node.

Structure of SNALink Components

The components of SNALink are:

- Local nodes
- SNALinks
- 3270 emulators

This section introduces the structure of these components and explains terms used to refer to the structure.

In This Section

Role of the Base

TheBase is a part of each Host Integration Server 2009 component, such as the local 2.1 node or a link service that provides the operating environment for that component. It passes messages between components and provides functions common to all components, such as diagnostic tracing.

The Link Base is the type of Base used by Host Integration Server SNALink. The Base has entry points for initialization, sending messages, receiving messages, and termination.

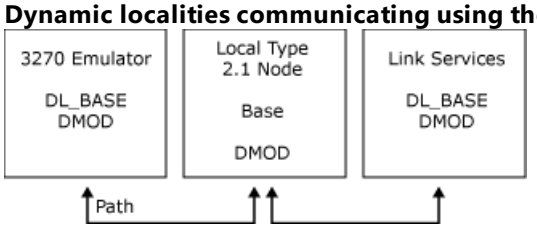
Localities and DMODs

A Base and its components (that is, a Host Integration Server 2009 executable program) is called a **locality**. The Host Integration Server system therefore consists of one or more communicating localities (all the running Host Integration Server 2009 executable programs within the LAN Manager domain). For each Host Integration Server system, there is a central configuration file. In addition, each Host Integration Server computer maintains configuration information about the SNALinks it supports.

In a system such as Host Integration Server, where the number of localities and their types are not configured in advance, the relationships between the localities are set up dynamically as individual localities come and go. Localities that can enter and leave a system in this way are called **dynamic localities**.

Dynamic localities communicate using the Dynamic Access Module (DMOD) component, which provides the communications facilities needed to pass messages between the Bases. This is illustrated in the following figure.

Dynamic localities communicating using the DMOD component



This figure shows a system consisting of three dynamic localities. Dynamic localities can enter or leave this system at any time.

Component Localities

SNALinks can enter dynamically into a Host Integration Server 2009 system. The SNALink, in conjunction with the Base, acts as a whole locality and communicates with the other localities in the system using a Dynamic Access Module (DMOD).

SNALink Interface describes the interface to the Base and the DMOD that allows a SNALink (or any other Host Integration Server component) to participate in a Host Integration Server system.

Partners

For Host Integration Server 2009 components and applications to communicate with each other, it must be possible to identify a partner within a locality. A partner is an addressable component of a locality; that is, code to which messages can be sent. In a Host Integration Server system, there is generally only one partner within a locality (such as an SNALink or the 3270 emulation program). However, separate functions within the local 2.1 node (such as the 3270 and Advanced Program-to-Program Communications (APPC) functions) can be considered to be separate partners.

SNALink Structure

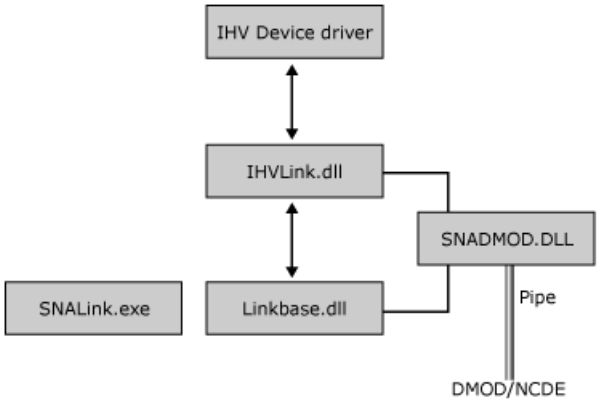
A Host Integration Server 2009 SNALink consists of the following:

- The link-specific protocol code provided by the independent hardware vendor (IHV)
- A Base
- A Dynamic Access Module (DMOD)

The DMOD, Base, and the IHV link-specific component of a Host Integration Server SNALink are implemented as dynamic-link libraries (DLLs). The executable component SNALINK.EXE is used to start a SNALink. SNALINK.EXE determines (from the Host Integration Server configuration) which link support DLL (for example, IHVLINK.DLL) is required and dynamically loads it before entering the Base scheduler.

The following figure shows the executable structure of a SNALink.

Executable structure of an SNALink



Messages

Messages are used to pass data between partners in the Host Integration Server 2009 system. This section provides information about message structure and formats.

In This Section

- Overview of Message Formats
- Buffer Header Format
- Buffer Element Format

Overview of Message Formats

A message always contains fixed-format header information such as a message type and addressing information. It can also contain other header information specific to a particular message type (such as the message subtype) and an indefinite amount of extra data.

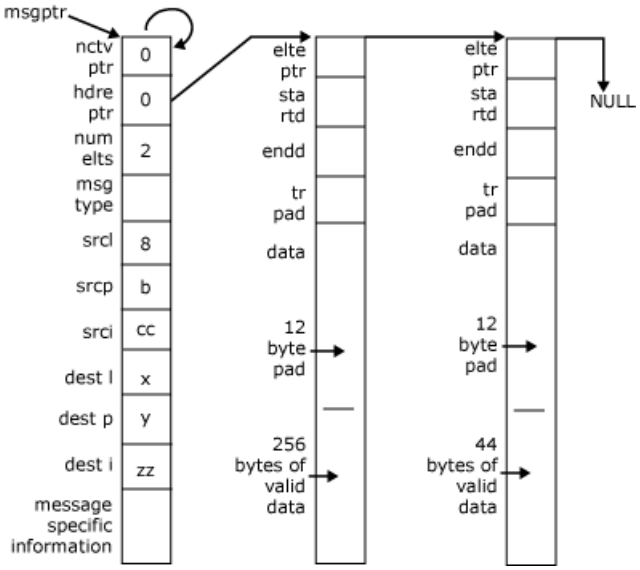
Messages are saved in buffers that consist of one header and zero or more elements:

- The header contains the fixed-format information and a pointer to an element. (This pointer will be NULL if there are no elements associated with the message.)
- An element contains any extra data for a message and a pointer to another element if the data continues into another element.

Buffer headers and elements are regarded as contiguous (8-bit) byte sequences. Messages of any length can be built up by chaining sufficient elements to a header.

The following figure shows a typical message with two elements.

Typical message with two elements



Buffer Header Format

This topic lists the common fields that always occur at the start of a buffer header. These are followed by further fields specific to the particular message. For details about individual message formats, see SNADIS Message Formats.

Field	Type	Description
PTRBFHDR	nxtqptr	When the buffer is in a queue, this field points to the header of the next buffer in the queue (NULL if it is the last buffer in the queue). When the buffer is not in a queue, this field points to itself. The Host Integration Server 2009 buffer management routines use this to check for buffer corruption.
PTRBFELT	hdrptr	Pointer to the first buffer element in the associated chain of buffer elements; NULL if the message consists only of a buffer header.
CHAR	numelts	Number of buffer elements chained from the header. Zero if the message consists only of a buffer header.
CHAR	msgtype	Message type. For more information, see individual message descriptions in SNADIS Message Formats.
CHAR	srcl	Source locality. For more information, see LPI Addresses.
CHAR	srcp	Source partner. For more information, see LPI Addresses.
INTEGER	srci	Source index. For more information, see LPI Addresses.
CHAR	destl	Destination locality. For more information, see LPI Addresses.
CHAR	destp	Destination partner. For more information, see LPI Addresses.
INTEGER	desti	Destination index. For more information, see LPI Addresses.

Note

Fields that occupy two bytes, such as **opresid** in the Open(LINK) request, are normally represented with the arithmetically most significant byte in the lowest byte address, irrespective of the normal orientation used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. However, the following fields are exceptions:

- The **srci** and **desti** fields in buffer headers are stored in the local format of the application that assigns them (only the assigning application needs to interpret these values).
- The **startd** and **endd** fields in elements are always stored in low-byte, high-byte orientation (the normal orientation of an Intel processor).

Buffer Element Format

This topic lists the common fields that always occur at the start of a buffer element. The **dataru** field contains information specific to the particular message. For details about individual message formats, see SNADIS Message Formats.

Field	Type	Description
PTRBFELT	hdrptr–>elteptr	Pointer to next buffer element in the chain. NULL if this element is the last or only element in the chain.
INTEGER	hdrptr–>startd	Start of valid data in this element. The index into *dataru* of the first byte of valid data.
INTEGER	hdrptr–>endd	End of valid data in this element. The index into *dataru* of the last byte of valid data.
CHAR	hdrptr–>trpad	Pad byte (reserved).
CHAR[SNANBEDA]	hdrptr–>dataru	An array of characters that contains the data for this element. Note that the valid data might not occupy the whole of the element. The **startd** and **endd** fields give the indexes into this array of the start and end of the valid data. The constant SNANBEDA is defined in SNA_DLC.H as 268.

The following information will help you to interpret the message formats:

- Fields that occupy two bytes are represented with the arithmetically most significant byte in the lowest byte address, irrespective of the normal orientation used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. The exceptions to this are the **startd** and **endd** fields in elements, which are always stored in low-byte, high-byte orientation (the normal orientation of an Intel processor).

- The offsets indicated by the **startd** and **endd** fields are expressed in terms of the first byte of dataru being offset 1; the first byte of valid data is at **dataru(startd–1)**. For example, if **startd** is 11 and **endd** is 18, **dataru** begins with 10 bytes that are not valid data, followed by 8 bytes of valid data.

In the example message format illustrated in Overview of Message Formats, each element has a **startd** of 13, indicating 12 bytes of padding before the start of the valid data. This leaves room for 256 bytes of data, and hence the element data (300 bytes long in this example) requires two elements.

LPI Connections

Partners communicate by passing messages to each other. If two partners want to communicate with each other, a locality, partner, index (LPI) connection is set up between the two partners. Messages then flow between the partners over this connection. The term *LPI connection* is explained in LPI Addresses. Note that this is not related to the Microsoft Host Integration Server 2009 concept of a connection between the local node and a remote system.

In This Section

- Paths and DMODs
- LPI Addresses
- Making Connections

Paths and DMODs

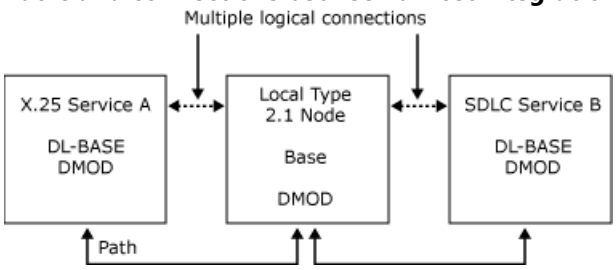
Dynamic Access Modules (DMODs) are responsible for the communication between localities. When the DMODs in two localities can successfully pass messages between them, a path is said to exist between the two localities. A path must exist between two localities before a connection can exist between partners in those localities.

In Host Integration Server 2009, a path is implemented using reliable LAN connections (named pipes, SPX, TCP, VINES IP)—one LAN connection for each path. When the two localities are on the same computer, a local pipe is used. This is implemented using shared buffers to increase performance, but is used by the application in exactly the same way as communication with a remote locality.

The DMOD provides communication between dynamic localities and provides guaranteed in-order delivery of messages flowing over paths between localities. If the DMOD loses its path to another locality, it informs the Base.

The following figure illustrates the paths and connections between a Host Integration Server local node and two SNALinks. X.25 service A has two connections to the local node (one for each of two virtual circuits); SDLC service B has one connection to the local node.

Paths and connections between a Host Integration Server local node and two SNALinks



LPI Addresses

A locality, partner, index (LPI) address is used to identify each end of a connection. It has three components: locality (L), partner (P), and index (I).

- **Locality** is a 1-byte identifier that uniquely identifies a locality within a system. This locality corresponds to a Host Integration Server 2009 component (local node, SNALink, 3270 emulator, and so on).

- **Partner** is a 1-byte identifier for the type of service. Each type of service has a unique value. A Host Integration Server local type 2.1 node has a defined value of 0x11. A Host Integration Server emulator has a defined product identifier of 0x12. A Host Integration Server link service (X.25. SDLC, Token Ring, Ethernet, or Channel, for example) has a defined value of 0x16.

- **Index** is a 2-byte identifier that uniquely identifies a logical entity within the product. The meaning and use of this field is defined by the communicating services. It is used to distinguish multiple connections between the same services (for example, to identify one of many virtual circuits available from an X.25 SNALink). The value of zero should not be used as an index. Applications must assign unique index values for every active LPI connection within the node.

A message flowing over a connection carries a pair of LPIs, identifying the source and destination of the message. These are the source LPI and destination LPI of the message. Together they identify the connection on which the message is flowing.

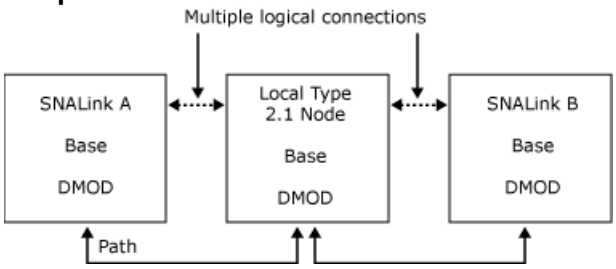
Note that more than one connection can exist between any pair of services. The Index values are then used to distinguish the connections. For example, in communications between the local node and a SNALink, the L and P values identify the message as being data link control (DLC) data for that local node, and the I value indicates which connection the data is intended for.

The LPIs are assigned by a combination of the products and the Dynamic Access Modules (DMODs) when the connection is opened, as described in Making Connections.

Because they are assigned dynamically for each component, the L values are not the same across a whole system. For example, a local 2.1 node locality could be known as locality 4 to one SNALink locality and locality 6 to a second SNALink locality. However, from the viewpoint of any locality, there exists a unique L value for each remote locality within which a path exists. This L value is used as an index into an internal table that identifies the path to that locality.

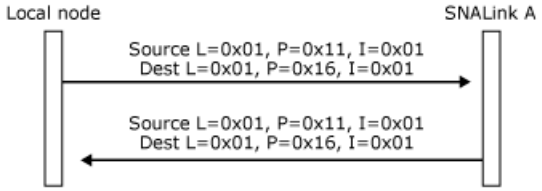
The following three figures show an example of the L values that could be used between the components shown in Paths and DMODs, and examples of the LPI values that would be used by the local node on messages flowing between the components.

Sample L values



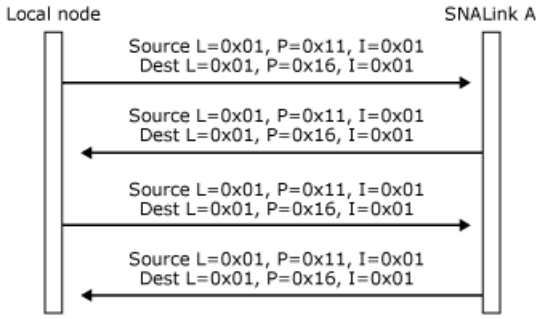
The following figure shows L values specified on messages between the local node and SNALink A.

L values specified on messages between the local node and SNALink A



The following figure shows LPI values specified on messages flowing on two different connections between the local node and SNALink A.

LPI values specified on messages flowing on two different connections



The Base is called by any piece of code that wants to send a message. It uses the destination L value on the message to determine where to send it. When the message gets to the remote locality, the Base in that locality routes it to the appropriate service if the locality contains more than one service.

Making Connections

Before messages can flow across connections, the connections must be established, or opened. This is necessary because a service does not initially know the locality, partner, index (LPI) address of the service with which it wants to communicate. There may not even be a suitable service for it to communicate with.

When a local node wants to communicate with an SNALink, it attempts to open a connection by sending an **Open(LINK)** request to the SNALink. This message will have LPI values already set up by the Base, which the SNALink should save for referencing the connection in the future.

The data link control (DLC) interface does not permit the SNALink to issue an Open request.

SNALink Interface

The SNALink interface specifies how an Independent Hardware Vendor (IHV) link DLL fits into the SNA link service architecture provided by the Base/DMOD interface. This section describes the SNALink interface, the entry points that an IHV link DLL can call, and those functions that a link service must provide to the Base/DMOD interface. These entry points allow messages to be sent to and received from the local 2.1 node.

In This Section

Process Structure and Scheduling

The primary thread of execution within a Host Integration Server 2009 SNALink is under the complete control of the Base. The Base schedules the SNALink by calling predefined entry points, which the IHV link support code must provide.

The IHV link support code can spawn extra threads of execution; however, the Base is not reentrant. The IHV code must ensure that only a single thread is executing within the Base at any moment in time.

The recommended SNALink structure uses the dispatcher to handle messages received from the local node and the work manager to process data received from the link. These routines and the way in which they are scheduled are described in The Dispatcher and The Work Manager respectively.

SNALink Initialization

When the SNALink is loaded into memory, the Base/DMOD performs all initialization required by the Host Integration Server 2009 system, including announcing availability of the new SNALink to other Host Integration Server components.

When this has been completed, the Base/DMOD calls the SNALinkInitialize function, which must be provided by the IHV link support code.

SNALinkInitialize is called with a parameter that is a handle to the global Base event. This handle should be saved by the SNALink and used to signal the Base when an event occurs (for example, when data is received from the link).

The **SNALinkInitialize** function should also:

- Read in the Host Integration Server configuration information for the SNALink. For details, see SNALink Configuration Information.
- Set up any required data structures.
- Register with the driver that provides the support for the hardware adapter, initializing this if necessary.

If initialization fails for any reason (for example, if an associated driver is not installed), the function should report the failure to the administrator by calling **SNAReportStatus**.

SNALink Termination

When a critical error occurs, forcing abnormal termination of the SNALink, the IHV code must ensure that all active connections are cleanly terminated, using whatever protocols are appropriate for the link type in use. For example, an X.25 SNALink would send a CLEAR packet on all active VCs and possibly take down level 2.

This should be performed using the process detach facility of the Win32® API function **DLLEntryPoint** .

Sending Messages

The SNALink should build a message in a buffer, and then call the Base to send it. The message contains source and destination LPIs, which are set up when the connection is opened. For more information, see LPI Connections.

The SNALink can either obtain a new buffer to contain the message to be sent (using SNAGetBuffer) or reuse one in which it has previously received a message. The application is responsible for any buffer that it has obtained or in which it has received a message. It must either use (or reuse) the buffer to send a message or release it (using SNAReleaseBuffer). If a buffer to be reused does not contain the correct number of elements for the message to be sent, the application can obtain additional elements (using SNAGetElement) or release existing ones (using SNAReleaseElement). It is the applications responsibility to maintain the **numelts** field in the message header.

The function used to send a message to the node is SNASendMessage.

Dispatcher

Whenever a Base event occurs, the Base calls the link support code dispatcher function SNALinkDispatchProc to handle the event. The term Base event in this context means:

- A message arriving from a local 2.1 node.
- A Base timer tick occurring—this relatively slow event happens approximately every five seconds.
- Losing contact with a local 2.1 node (for example, the machine being powered down).

The **SNALinkDispatchProc** function should examine parameters passed to it by the Base to determine why it has been called (for details, see Sample Code for SNALinkDispatchProc) and call an appropriate function to handle the event. When the event has been processed, control returns to the Base scheduler.

Receiving Messages

The Base calls the SNALink dispatcher function SNALinkDispatchProc when a message is available for it.

Note that after the application receives a message, it is responsible for the buffer in which the message was received. It must either reuse the buffer to send a message (using SNASendMessage) or release it (using SNAReleaseBuffer). If the buffer to be reused does not contain the correct number of elements for the message to be sent, the application can obtain additional elements (using SNAGetElement) or release existing ones (using SNAReleaseElement).

Work Manager

When no work is currently outstanding, the Base thread of execution sleeps, waiting for an event or for a maximum period of five seconds. SNALinks should signal the Base when an event occurs (such as data arriving on the link) by setting the Base global event. A handle to this event is passed on the SNALinkInitialize call.

When the Base is rescheduled, it calls the SNALink work manager function SNALinkWorkProc. This function should handle any link events that have occurred.

A common use of this function is in an SNALink where there is a single thread that handles the protocol of the link and also multiple threads suspended on synchronous calls to a driver read function. When data is received from the link, it is placed on an internal queue, and the driver sets the global Base event. This causes the Base to be scheduled, and **SNALinkWorkProc** is called. **SNALinkWorkProc** then removes messages from the queues and passes them to the Base to be sent to the local node.

Base/DMOD and SNALink Entry Point Summary

The following tables show entry points divided into the categories SNALink, buffer management, and Base/DMOD, and listed in alphabetic order within each category.

SNALink entry points

Entry point	Description
SNALinkDispatchProc	Dispatcher.
SNALinkInitialize	Initialize SNALink.
SNALinkTerminate	Terminate SNALink.
SNALinkWorkProc	Work manager.

Buffer management entry points

Entry point	Description
SNAGetBuffer	Get buffer.
SNAGetElement	Get buffer element.
SNAReleaseBuffer	Release buffer.
SNAReleaseElement	Release buffer element.

Base/DMOD entry points

Entry point	Description
SNAGetLinkName	Get the name of the SNALink.
SNASendAlert	Send a preformatted NMVT alert to NetView.
SNASendMessage	Send a message to the node.

The following functions are defined in SNALink Configuration Information:

Entry point	Description
SNAGetConfigValue	Get a named item of configuration information.
SNAGetSystemInfo	Get Host Integration Server 2009 system information.

Note
Standard calling conventions [WINAPI] are used for all entry points including those provided by the IHV SNALink.

The format of buffer headers and elements is described in Messages. The formats of individual messages contained in buffers are defined in SNADIS Message Formats.

Sample Code for SNALinkDispatchProc

This section contains outline source code for the link dispatcher function SNALinkDispatchProc.

```
/*********************************************************************/
/* First, include the SNA service header files                       */
/*********************************************************************/
#include <sna_dlc.h>
#include <sna_cnst.h>
#include <trace.h>

/*********************************************************************/
/* The link dispatcher routine - SNALinkDispatchProc                 */
/*********************************************************************/
VOID SNALinkDispatchProc (msgptr, function, locality)
PTRBFHDR msgptr;
INTEGER  function;
INTEGER  locality;
{
  INTEGER     discard_buff;
  COM_ENTRY("Ldisp");
  if (msgptr != NULL)
  {
    TRACE4()"received message from local node"));
    discard_buff = FALSE;
    switch (msgptr->msgtype)
    {
      case OPENMSG:
             /* process the OPEN message */
             break;
      case CLOSEMSG:
             /* process the CLOSE message */
             break;
      case DLCDATA:
             /* Data to be sent on link */
             break;
      case DLCSTAT:
             /* Switch on the subtype of the message */
             switch (msgptr->dshdr.dstype)
             {
               case STRESRCE :
                      /* call flow control processor */
                      break;
               case DLCSDXID:
                      /* call XID processor */
                      break;
               default:
                      discard_buff = TRUE;
                      break;
             }
             break;
      default:
             discard_buff = TRUE;
             break;
    }
    if (discard_buff)
    {
      /* message has not been processed, so simply discard */
      SNAReleaseBuffer(msgptr);
      msgptr = NULL;
    }
  }
  else if (function == SBLOST)
  {
    /* Lost contact with local node 'locality'  */
    /* Terminate all connections on this node (matching destl-value) */
```

```
    }
    else if (function == SBTICK)
    {
      /* 5 second timer tick */
    }
    COM_EXIT;
  }
```

SNALink Configuration Information

The configuration information for all SNALinks on a computer is stored hierarchically, referenced by the SNALink name.

The entry for each SNALink must include certain fields that are required by the Host Integration Server 2009 system. These are listed in the following table.

Required field	Description
TYPE	The type of the SNALink. Acceptable values for TYPE are: SDLC, X25, TOKENRING, TCPIP, FRAMERELAY, CHANNEL, ISDN, ETHERNET.
LINKMODULE	The name of the IHV DLL that provides the protocol code.

The remainder of the configuration information consists of entries of the form PARAMETER = VALUE. Parameters can be set to either an integer or a string.

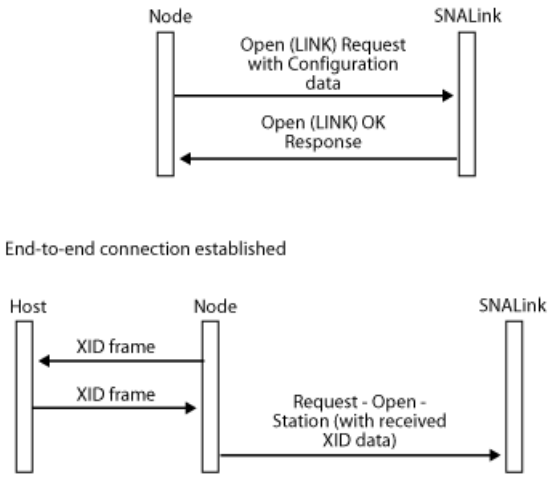
Examples of possible parameters that may be required by an SNALink are as follows:

- PortNumber = 3
- LineType = SWITCHED
- L3PacketSize = 128
- T1Timeout = 30

Note that to support more than one port on a multiport adapter, you must define multiple SNALinks. It is not possible to configure a single SNALink to support more than one physical link.

The following figure shows a sample configuration for a computer with two SNALinks—SDLC1 and X25HOST.

Sample configuration for a computer with two SNALinks



The configuration information is accessed using API calls that Host Integration Server provides.

The IHV Setup utility must write the configuration information for each SNALink supported. See Setup Information for information about how this should be performed.

In This Section

Accessing Configuration Information

Accessing Configuration Information

The following table lists the calls the SNALink uses to obtain its configuration information.

Call	Description
SNAGetConfigValue	Returns the value of a named configuration parameter.
SNAGetSystemInfo	Returns general information on the version of SNA server currently running, such as the release level, and the network operating system.

If the return code from **SNAGetConfigValue** indicates that the specified configuration parameter is not available, or if the information returned is invalid, it is the SNALink's responsibility to decide what action to take. If appropriate, an error message could be logged.

It is strongly recommended that the SNALink read all required configuration parameters at initialization time (when SNALinkInitialize is called by the Base). This safeguards against the configuration information changing while the link service is running.

Note
Standard calling conventions (WINAPI) are used for all entry points, including those provided by the IHV SNALink.

Data Link Control Interface

The data link control (DLC) interface defines the interface between the local 2.1 node and an SNALink. The DLC interface is defined in terms of the messages that are sent across the interface. Note that this is logically distinct from the definition of the Base/Dynamic Access Module (DMOD) interface, which defines the API used to send messages between two components in Host Integration Server 2009 (for example, between the local node and an SNALink).

DLC messages are exchanged between the local node and an SNALink across LPI connections. For details, see Structure of SNALink Components.

The local node uses the DLC interface to:

- Activate DLC connections.
- Exchange format 0 or format 3 XIDs for station activation.
- Exchange DLC information frames.
- Handle DLC error notification.

In This Section

Supported Configurations

The 2.1 node supports the full range of station roles:

- Primary data link control (DLC) stations
- Secondary DLC stations
- DLC station role negotiation

For Synchronous Data Link Control (SDLC) connections, the node allows:

- Leased lines configured as:
    - Secondary point-to-point
    - Primary point-to-point or multipoint
    - Negotiable point-to-point
- Switched lines (point-to-point only) with:
    - Remote physical unit (PU) identification through XID exchange
    - Auto dial (with suitable hardware support)
    - Incoming call support

For X.25 and 802.2, the node also supports:

- Multiple connections over one physical link
- Incoming calls with validation of caller's address

In addition, for X.25, the node supports permanent virtual circuits (PVCs) and switched virtual circuits (SVCs).

Opening a Connection

The 2.1 node is capable of supporting multiple connections through one or more SNALinks. For each connection, the node opens two Locality Partner Index (LPI) connections to the SNALink:

- LINK LPI connection to handle activation and deactivation of the connection.
- STATION LPI connection to transfer data to and from the remote station.

The one exception to this rule is the case of primary multipoint connections where there is a single LINK LPI connection and multiple STATION LPI connections. This special case is described in SDLC Multipoint Connections.

The following messages flow over the data link control (DLC) interface and are used to activate a connection to a remote station.

Message	Description
Open(LINK) Request	• Flows from node to DLC over LINK connection. • Opens the LINK LPI connection between the node and the SNALink. • Provides configuration data for the SNALink. • Provides link connection data such as Token Ring address for the remote station.
Open(LINK) Response	• Flows from DLC to node over LINK connection. • Reports whether the SNALink has accepted the **Open(LINK) Request**. • Returns certain link-specific configuration parameters to the local node. • Can be an OK Response or an Error Response.
Request-Open-Station	• Flows from DLC to node over LINK connection. • Passes an XID received from the SNALink up to the node. • Indicates that the SNALink has received a mode setting command, such as SNRM over S DLC, or SABME over 802.2.
Send-XID	• Flows from node to DLC over LINK connection. • Passes an XID from the node to the SNALink to be sent out over the link to the remote s tation.
Open(STATION) Request	• Flows from node to DLC over STATION connection. • Opens the STATION LPI connection between the node and the SNALink. • Specifies certain station-specific configuration information.

Open(STATION) OK Response –or– Open(STATION) Error Response	• Flows from DLC to node over STATION connection. • Acknowledges **Open(STATION) Request**.
Station-Contacted	• Flows from DLC to node over STATION connection. • Informs the local node that the link is now ready for data transfer.

The use of these messages in activating various types of connections is described throughout the rest of this section. For information about the format of the messages, see SNADIS Message Formats.

The name of the Request-Open-Station message is historical. In earlier versions of the DLC interface, the higher-level software (such as the local node) always sent an **Open(STATION) Request** in response to this message—hence the name Request-Open-Station. However, now that multiple XIDs can be exchanged before the link is activated, the **Open(STATION) Request** is only sent at the end of the XID exchange.

The Request-Open-Station message now has two distinct semantic meanings:

- A Receive-XID
- A Receive-Set-Mode

In This Section

- Opening the LINK LPI Connection
- Activating a Host Connection
- Activating a Peer Connection
- Opening the STATION LPI Connection
- Node Identification and Signaling Information
- XID Retries
- Multiple Connections

Opening the LINLPI Connection

The local node attempts to activate a connection:

- During system initialization if the connection is configured as initially active.
- If the system administrator manually activates the connection.
- If a 3270 or logical unit (LU) 6.2 session is requested when there is no active connection to support the LU, and the connection is configured to be activated on demand.

For each connection to be activated, the local node opens a LINK Locality Partner Index (LPI) connection by sending an Open(LINK) Request to the SNALink. This message contains configuration data such as:

- Synchronous Data Link Control (SDLC) line type: leased, switched.
- Operational role: primary, secondary, or negotiable.
- Time-out values.
- Retry limit values.
- Line speed.
- Half-duplex/full-duplex.
- 802.2 remote service access point (SAP) address.
- X.25 facility data.

For incoming calls, the local node primes the SNALink by opening the LINK LPI connection, but does not perform an activation sequence at this stage. For details, see Incoming Call Support.

The local node also inserts the first XID frame to be used (where applicable) and link connection data to be used on a switched link. The link connection data can be:

- A telephone number for a manual or autodial modem (in this case, the SNALink software could dial the required number or send a message to the operator specifying the number to be dialed).
- The media access control (MAC) address of the remote station.
- The X.25 remote data terminal equipment (DTE) address.

Finally, the **Open(LINK)** contains various time-out values that should be used by the SNALink when setting up protocol timers. For more information, see Open(LINK) Request and Open(LINK) Response.

The SNALink should return an **Open(LINK) OK Response** if:

- Its internal control blocks are successfully initialized.
- Its device driver has installed correctly.
- Its link hardware is successfully initialized.

The SNALink should not wait for an end-to-end connection before giving an **Open(LINK) Response**.

If the SNALink has successfully initialized, it should return an **Open(LINK) OK Response** immediately, supplying the required link-specific configuration information to the node (such as the maximum BTU size it can support). The local node uses this information during XID negotiation with the remote station.

If the SNALink cannot initialize successfully, it responds with an **Open(LINK) Error Response** containing an error code. The error is logged and the local node notifies the system operator before retrying the link activation.

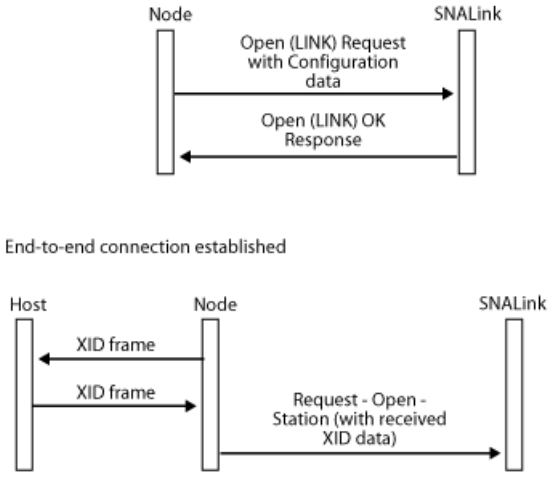
If an XID is supplied on the **Open(LINK) Request**, this should be sent when the end-to-end connection is established for a primary or negotiable link. Note that the supplied XID can be a NULL XID, which has a zero length. Hence, it is important that the XID field is examined rather than checking for a zero XID length. An XID will be supplied for all connections except primary leased connections (which could be multipoint).

When an SNALink receives an XID frame from the remote station, it is passed to the local node in a Request-Open-Station message on the LINK LPI connection.

If the SNALink fails to receive any frames from the remote station, it generates an Outage message as described in Closing a Connection.

The following figure shows the **Open(LINK) Request** and **Open(LINK) Response**, followed by an exchange of XIDs.

Open(LINK) Request and Open(LINK) Response, followed by an exchange of XIDs



Activating a Host Connection

A host connection can be activated over a leased synchronous data link control (SDLC) line, X.25, 802.2, or a switched SDLC line. This section describes the activation procedures for each type of connection.

In This Section

Leased SDLC Line (No XIDs Exchanged), Channel Adapter

For a connection to a host computer using a leased synchronous data link control (SDLC) line, the SNALink receives a set normal response mode (SNRM) when the end-to-end connection is established. The SNALink responds with an unnumbered acknowledgement (UA) and informs the local node that the connection is ready for data transfer. This is done with the Request-Open-Station message with the Rcv-Set-Mode flag set.

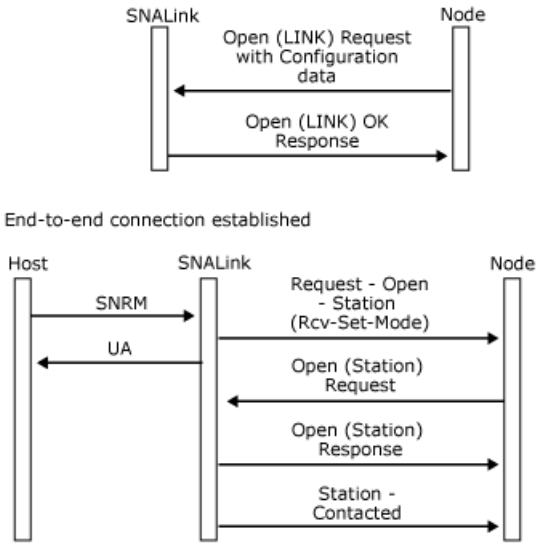
The node then opens the STATION Locality Partner Index (LPI) connection with the Open(STATION) message. If the SNALink has an available control block, it responds with an **Open(STATION) OK Response**. This is followed by a Station-Contacted message.

A channel connection is treated the same way as a leased secondary SDLC connection. Each channel connection is associated with a channel subaddress in the range 0x00 to 0xFF. The SNA service node sends the channel link service an Open(LINK) Request for each configured channel connection when the connection is activated. The link service should expect to receive multiple Open(LINK) Requests, one for each supported subchannel address.

Note that the Request-Open-Station message flows on the LINK LPI connection, whereas the Station-Contacted message flows on the STATION LPI connection.

The message flow for a leased line is shown in the following figure.

Message flow for a leased line



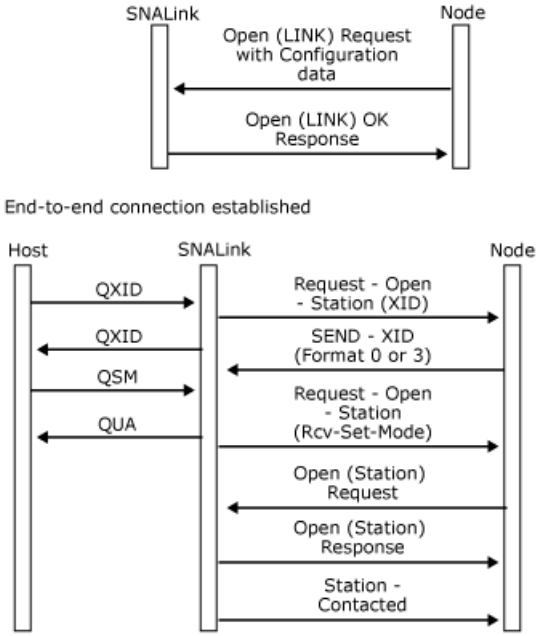
X.25, 802.2, or Switched SDLC Line (XIDs Exchanged)

The initial sequence for a host connection over X.25, 802.2, or a switched synchronous data link control (SDLC) line is similar to the sequence over a leased line. The only difference is that exchange identifications (XIDs) are exchanged before the host (or front-end processor) sends a mode-setting command such as set mode (QSM) on an X.25 qualified logical link controlQLLC) link.

When the SNALink receives an XID, it is passed to the local node on a Request-Open-Station message (on the LINK Locality Partner Index (LPI) connection). The local node then passes the data link control (DLC) a Send-XID message (also on the LINK LPI connection) containing the XID to be sent to the host. The host typically checks the node identifier in this XID and, if it is valid, sends the mode-setting command.

The sequence is shown in the following figure.

Sequence for passing an XID



For switched connections using SDLC modems, the Open(LINK) Request contains dial digits for manual or auto-dial modems. It is the responsibility of the SNALink to handle the management of these devices. For X.25 and 802.2 connections, the **Open(LINK) Request** contains the address of the remote station.

The SNALink should initiate the dialing procedure when it receives the **Open(LINK) Request**.

Activating a Peer Connection

For a peer connection, there is an activation sequence that involves the two stations exchanging format 3 exchange identification (XID) frames. As part of this sequence, the two stations agree on their link roles. They also exchange information relating to the link level connection, such as the maximum frame size supported.

The node passes XIDs to the SNALink over the LINK LPI connection using the Send-XID message. The SNALink returns received XIDs to the local node over the LINK LPI connection using the Request-Open-Station message.

Fixed Link Roles and Negotiable Link Roles show examples of XID exchange for the two cases:

- The link roles are explicitly configured for the two stations.
- The link roles of both stations are negotiable.

Points to note are:

- The Open(LINK) Request is supplied with a NULL XID that is sent when the end-to-end connection is established.
- After the first NULL XID, all XIDs are format 3.
- If both stations are set up to be negotiable, the station with the higher node identifier becomes the primary.
- If both stations are negotiable and have the same node identifier, both stations produce randomized node identifiers that are compared as before.

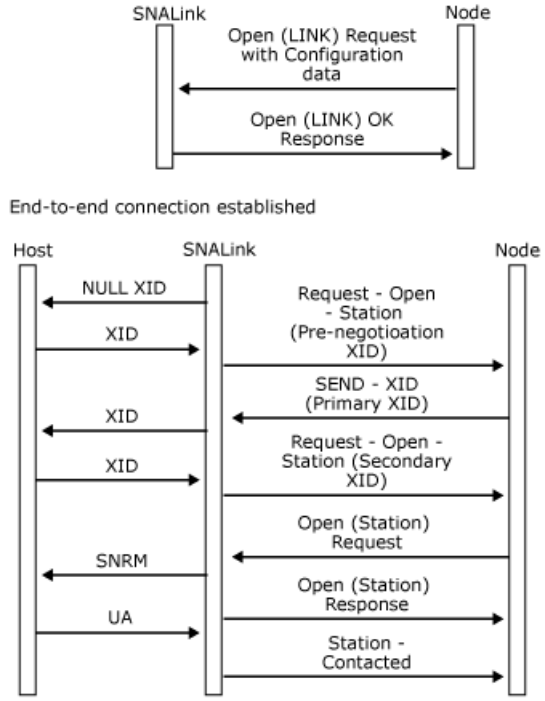
In This Section

- Fixed Link Roles
- Negotiable Link Roles

Fixed Link Roles

The following figure shows the sequence of messages for a peer connection where the local end is configured as the primary station and the remote end is configured as the secondary.

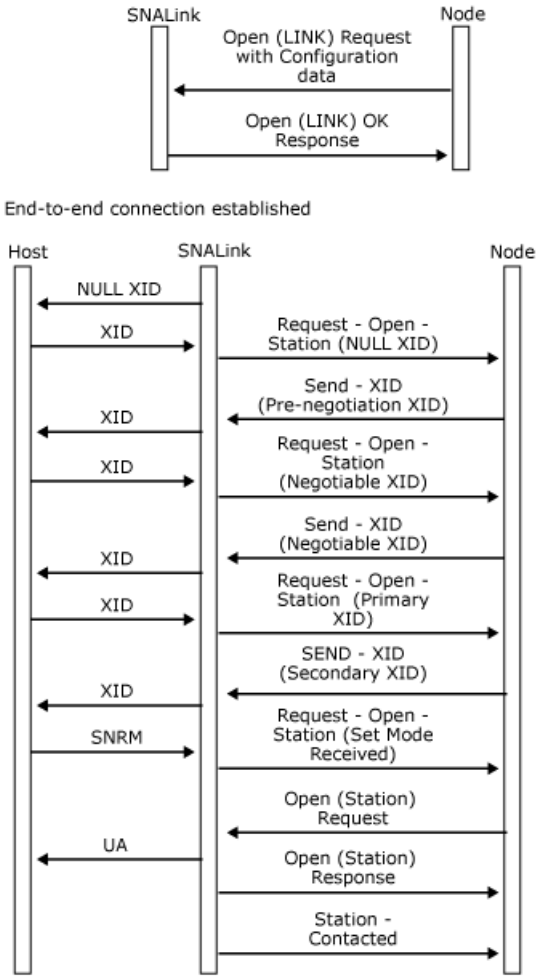
Sequence of messages from a peer connection where the local end is configured as the primary station



Negotiable Link Roles

The following figure shows the sequence of messages for a peer connection where both the local and remote ends are configured as negotiable. Because the remote node identifier is larger (numerically) than the local node identifier, the remote station will become primary.

Sequence of messages for a peer connection where both ends are configured as negotiable



local node identifier	= 0x05D11111
remote node identifier	= 0x05D22222

The following summaries the rules that the SNALink must follow when supporting exchange identification (XID) exchange, and in particular XID role negotiation:

- If an XID is supplied in the Open(LINK) Request, it must be transmitted as soon as the end-to-end connection is established for primary or negotiable links.
- All XIDs received from the remote station must be passed to the local node in a Request-Open-Station message.
- An XID received from the local node in a Send-XID message must be transmitted immediately.
- XID transmissions must be retried until an XID is received from the remote station. For half-duplex links, the retry time-out should be randomized to prevent repeated XID clashes.
- When a mode-setting command, such as set normal response mode (SNRM), QSM, or set asynchronous balanced mode extended (SABME), is received before the station has been opened, a Request-Open-Station must be sent to the local node with the Rcv-Set-Mode flag on.
- When the local node sends an Open(STATION) message, the link should examine it to determine its link role (that is, primary or secondary).

- A secondary station should send a Station-Contacted message after receiving and responding to the **Open(STATION)** message.

- For a primary station, the mode-setting command should be sent when the **Open(STATION)** message is received. The Station-Contacted message should be sent to the local node when this command has been acknowledged by the secondary station (for instance, an unnumbered acknowledgement (UA) received on an Synchronous Data Link Control (SDLC) link).

If the local node detects an error during role negotiation, such as both physical units (Pus) configured as primary, it sends out an XID containing an error vector. The vector is appended to the end of the normal XID data. The vector number specified is 0x22, and the vector data specifies that the data link control (DLC) role field is in error.

After sending the error XID, the local node sends a **Close(LINK)** message to terminate the connection (see Closing a Connection).

The following table is a matrix of the possible combinations of station link roles and shows the eventual role of the local station.

	Local Station			
Remote Station		**Primary**	**Secondary**	**Negotiable**
	Primary	Fail	Secondary	Secondary
	Secondary	Primary	Fail	Primary
	Negotiable	Primary	Secondary	Either*

*The station with the higher node identifier becomes the primary.

Opening the STATION LPI Connection

After receiving a Request-Open-Station (RQOS) message, the local node sends an Open(STATION) message to the SNALink when:

- The station is configured as secondary (or has negotiated to secondary), and the RQOS message has the Rcv-Set-Mode flag on, indicating that the SNALink has received a mode-setting command such as set normal response mode (SNRM).

- The station is configured as primary (or has negotiated to primary), the RQOS message contains a secondary exchange identification (XID), and the local station has sent at least one negotiation-proceeding XID.

The Open(STATION) Request contains the link index that was on the Open(LINK) Request. This field is used to correlate the LINK and STATION LPI connections for SNALinks that support multiple connections, such as X.25, 802.2, and channel.

This **Open(STATION) Request** contains configuration data for the link station, such as the Synchronous Data Link Control (SDLC) address of the adjacent link station (0x00 for secondary stations, 0x01 to 0xFE for primary stations).

The SNALink should use this address field for determining the local station's role. If it is set to 0x00, the local station is a secondary station. Otherwise, the local station is the primary station. This field has this meaning even when the address is not used because of the link type (such as 802.2).

See Open(STATION) Request for the format of this message.

If the station cannot be initialized (perhaps due to a lack of resources), the SNALink responds with an Open(STATION) Error Response containing the appropriate error code.

Node Identification and Signaling Information

For information about the role an SNALink plays in node identification, see Incoming Call Support.

When exchange identifications (XIDs) are exchanged, there are two mechanisms for identifying the remote station:

- The node identifier on received XIDs.
- The data link control (DLC) defined address; for example, the media access control (MAC) address. This is known as signaling information.

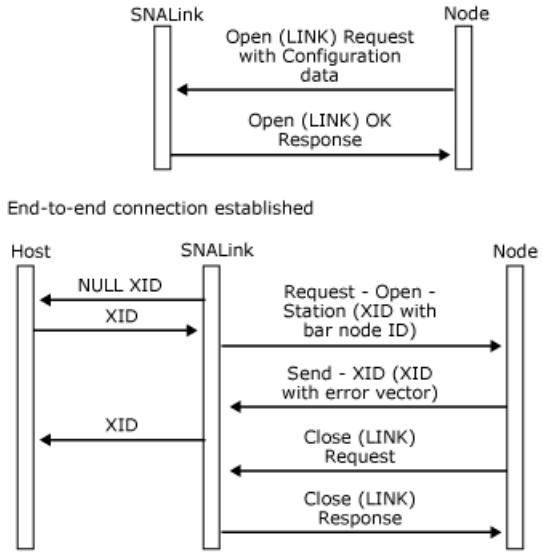
The presence of signaling information depends on the type of the SNALink. For instance, there is no signaling information over a Synchronous Data Link Control (SDLC) link, but there is signaling information over X.25 and 802.2. The SNALink passes signaling information to the local node on the Request-Open-Station message by appending it after the XID.

If signaling information is present, the local node checks it against the configured value in the dial-digits record of the Host Integration Server 2009 configuration file. For incoming call support, this allows the local node to determine the connection that is to be activated. For a fuller description of incoming calls, see Incoming Call Support.

If there is no signaling information, the local node compares the control point (CP) name on the received XID with the remote control point name in the configuration.

If the remote station is identified correctly, XID exchange proceeds as detailed in Activating a Peer Connection. However, if there is a mismatch, the local node sends an XID (in the Send-XID message) containing an error vector followed by a Close(LINK) Request, as shown in the following figure.

Local node sending an XID containing an error vector, followed by a Close(LINK) Request



XID Retries

When the local node specifies that the SNALink is to send an exchange identification (XID), either by supplying it on the Open(LINK) Request or by sending it on a Send-XID message, it is the responsibility of the SNALink to perform any retries.

XIDs need to be retried because:

- The remote station has not been started yet.
- Frames may be lost on the line due to noise.

Synchronous Data Link Control (SDLC) SNALinks should implement a contact time-out and a retry limit—values for these are provided on the Open(LINK) message. The time-out specifies how often the XID should be retried, and the retry limit specifies how many XIDs should be sent before abandoning the connection activation and sending an Outage message to the local node. The SNALink should stop retrying the XID when one of the following occurs:

- It receives an XID from the remote station.
- An Open(STATION) message is received from the node.
- A mode-setting command is received on the link.

Multiple Connections

For 802.2 and X.25 links, multiple connections can use the same physical link supported by a single instance of the SNALink software.

For each connection to a remote station, there is a LINK LPI connection and a STATION LPI connection (this is different from Synchronous Data Link Control (SDLC) multipoint as described in SDLC Multipoint Connections). Hence, there can be multiple pairs of LPI connections between a local node and an SNALink. For each connection, the local node issues an Open(LINK) Request and, after exchange identification (XID) exchange, an Open(STATION) Request.

The local node is configured with a maximum number of connections that can be active at any one time. In addition, each potential connection is configured with the address of the remote station. This information is required when activating a connection, and is included in the **Open(LINK) Request** for an outgoing connection.

DLC Information Transfer

When the local node has opened the LINK and STATION LPI connections and received a Station-Contacted message from the SNALink for a remote station, it can exchange data using the data link control (DLC) interface with the physical unit (PU) and associated logical units (LUs) at the remote station.

Data messages are contained within buffers. The transmission header (TH) of the SNA path information unit is contained within the buffer header. The request/response header (RH), if present, and request/response unit (RU) are contained within one or more buffer elements. The TH is contained in the buffer header for historical reasons. Typically, this will be copied by the SNALink into the element before **startd**, to keep the entire frame in contiguous memory locations. For a description of the DLC-Data message format, see DLC-Data.

Note that all data messages to a specified remote station flow on the associated DLC STATION LPI connection, and not on the controlling DLC LINK LPI connection.

In This Section

- DLC Flow Control

DLC Flow Control

The flow of data messages at the data link control (DLC) interface for each link station is flow controlled. For each direction of flow, there is an initial credit of messages that can be transmitted.

Flow control is maintained by initial specification on the Open(STATION) Request and Open(STATION) OK Response messages, and by the sending of DLC Status-Resource messages to give more credit periodically.

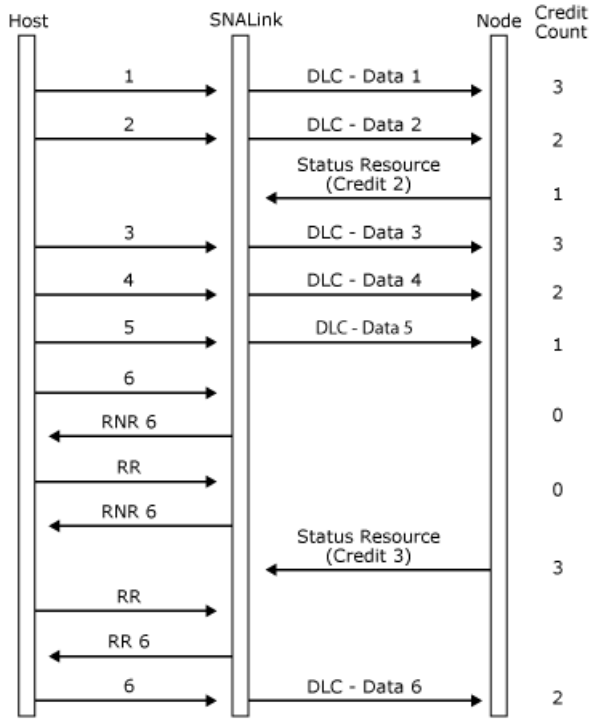
The sender maintains a count of credit, starting at the initial value set on the **Open(STATION)**, which is decremented for each DLC-Data message sent. When the credit count reaches zero, no more DLC-Data messages can be sent until more credit is received.

For flow in a given direction, the amount of credit is specified by the recipient of the data, because the recipient has to do any queuing. The initial credit values are passed on the Open(STATION) message (on the request for flow from the SNALink to the local node and on the response for flow from the local node to the SNALink).

The initial credit for the flow from the SNALink to the local node is determined by the node. The initial credit for the flow from the local node to the SNALink is set by the SNALink software—a suggested value is 16.

If the SNALink runs out of credit to send to the local node, it should either queue the data or discard it and send no acknowledgment. It should also start sending receive not ready (RNR), for example, when polled by a primary station. An example message flow with an Synchronous Data Link Control (SDLC) SNALink is shown in the following figure with an initial credit of 3. When the SNALink runs out of credit, it does not acknowledge any further frames and starts sending RNR.

Message flow with an SDLC SNALink with an initial credit of 3

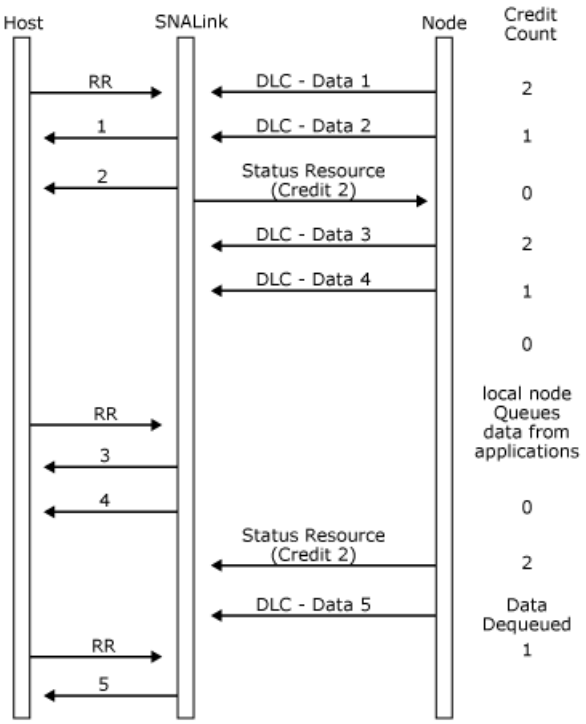


For flow control from the local node to the SNALink, when the node runs out of credit, it queues the data and applies back pressure on sessions using that station. There is thus end-to-end flow control in this direction, independent of any SNA pacing that may be in force.

The SNALink gives credit to the local node for the messages that have been transmitted, not for the messages for which acknowledgments have been sent. The amount of data queuing in the SNALink is kept down most of the time because frames will usually be acknowledged.

Flow control for the flow of data from the local node to the SNALink is shown in the following figure, where the initial credit is assumed to be 2.

Flow control for the flow of data from the local node to the SNALink


Host
SNALink
Node
Credit Count
RR
DLC - Data 1
2
1
DLC - Data 2
1
2
Status Resource (Credit 2)
0
DLC - Data 3
2
DLC - Data 4
1
0
RR
local node Queues data from applications
3
4
0
Status Resource (Credit 2)
2
DLC - Data 5
Data Dequeued
RR
1
5

Closing a Connection

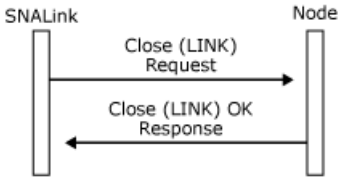
The local node closes a connection to a remote station:

- If the system administrator manually deactivates the connection.
- If the connection is configured as on-demand and no 3270 or LU 6.2 sessions are active.
- If an outage has been reported by the SNALink.

The local node closes a connection by sending a Close(LINK) message. The SNALink then takes some action, such as lowering Data Terminal Ready (DTR) on an Synchronous Data Link Control (SDLC) link or issuing a DLC_CLOSE_STATION on a 802.2 connection. It then replies with a **Close(LINK) OK Response** as shown in the following figure.

The case of multipoint connections is slightly different and is considered in SDLC Multipoint Connections. The following topics discuss point-to-point connections.

Local node receiving a Close(LINK) and replying with a Close(LINK) OK Response



In This Section

- Outages
- Connection Retries

Outages

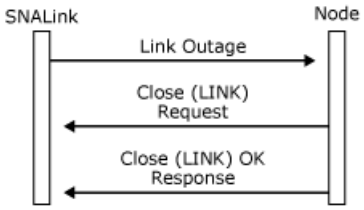
If the SNALink detects a link or station failure, it reports the failure by sending an Outage message to the node on either the LINK or STATION LPI connection depending on whether it is a link or station outage. Generally, a station outage indicates a problem at the remote station, and a link outage indicates a local or line problem.

When the local node receives an Outage message, it:

- Logs an error containing the outage code.
- Cleans up each session using the connection and informs applications of the failure (for instance, with a Comm Check code on a 3270 emulator).
- Sends a Close(LINK) Request to the SNALink.

On receipt of the **Close(LINK) Request**, the SNALink should clear up its internal resources for the connection and send back a Close(LINK) Response.

Local node receiving an Outage message and sending a Close(LINK) Request and a Close(LINK) response



There is a special case when the node loses contact with the SNALink software. In this case, the node is notified of this event (a lost locality) and performs outage processing apart from sending messages to the SNALink.

The outage codes are not distinguished by the node, but they are logged. For the sake of consistency across SNALink implementations, the values listed in the following topics should be used.

In This Section

- SDLC Outage Codes
- 802.2 Outage Codes
- X.25 Outage Codes

SDLC Outage Codes

The following table describes Synchronous Data Link Control (SDLC) outage codes.

Outage code	Description
0x0D	Internally generated for SNALink lost locality.
0x11	Data set ready (DSR) failure.
0x12	Clear to send (CTS) failure.
0x14	Data carrier detect (DCD) failure.
0x24	Nonproductive receive retry limit exceeded.
0x25	Idle time-out retry limit exceeded.
0x29	Connection problem. subqual = 0x00I-frame retransmission subqual nonzeroXID retransmission
0x2D	Abnormal modem response.
0x2E	Write time-out retry exceeded.
0xA0	Exchange identification (XID) exchange failed on multidrop line. Subqual is address of secondary station.
0x15	Discontact (DISC) received.
0x23	Receive buffer overrun.
0x2C	Invalid command received. subqual = 0x03invalid N(R) subqual = 0x04invalid or unsupported command/response s ubqual = 0x05excess I-field
0x80	Disconnect mode (DM) received in information transfer state.
0x81	Discontact retry limit exceeded.
0x82	Contact retry limit exceeded.
0x83	Poll retry limit exceeded.
0x84	No response retry limit exceeded.
0x85	Remote busy retry limit exceeded.
0x86	Frame reject (FRMR) received. subqual = 0x00no reason given subqual = 0x03invalid N(R) subqual = 0x04invalid or unsupported command/response subqual = 0x05excess I-field
0x87	Invalid frame received. subqual = 0x03invalid N(R) invalid or unsupported command/response subqual = 0x05exces s I-field
0x88	Request Initialization Mode (RIM) received.
0x89	Request Disconnect (RD) received.

802.2 Outage Codes

The following table describes 802.2 outage codes.

Outage code	Description
0x29	Remote node not active.
0xAB	The set asynchronous balanced mode extended (SABME) received while connection active.
0xAC	Frame reject (FRMR) sent.
0xAD	FRMR received.
0xAE	Discontact (DISC)/disconnect mode (DM) received.
0xAF	Link lost.

X.25 Outage Codes

The following table describes X.25 outage codes.

Outage code	Description
0x37	Loss of a virtual circuit.
0x60	Switched virtual circuit (SVC) cleared down by remote station or network.
0x61	Permanent virtual circuit (PVC) has been reset by remote station or network.
0x62	Attempt to connect to remote station through SVC failed.

Connection Retries

If an initially active or operator-started connection is closed because of an outage, the node periodically tries to reopen the connection. This can be stopped by manually deactivating the connection. This retry mechanism is also used when the node attempts to open a connection but the SNALink software has not yet been started. On-demand connections are not retried automatically by the node, but will be retried if the user attempts to reactivate a session using the connection.

Note that this connection retry timer is totally separate from the timers specified on the Open(LINK) Request.

Incoming Call Support

The local node allows an SNALink to be set up to support incoming calls. In this mode of operation, the node primes the SNALink by sending an Open(LINK) Request, but the SNALink does not attempt to activate the link until it receives an exchange identification (XID) from a remote station.

The SNALink recognizes an **Open(LINK) Request** for an incoming call by the absence of a connection name in the destination name field (this field is filled with ASCII blanks).

For incoming calls, **Open(LINK) Request** requires an immediate response from the SNALink, just as in the case of an **Open(LINK) Request** for an outgoing call.

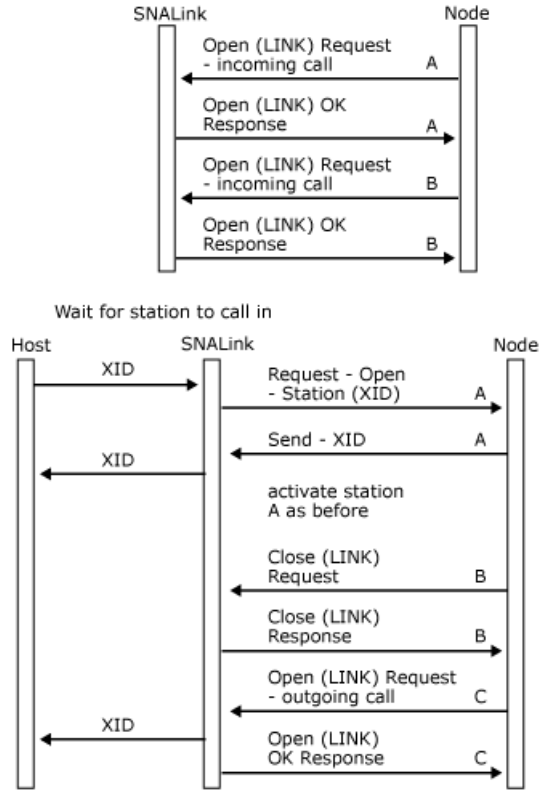
For a Synchronous Data Link Control (SDLC) SNALink, there can be only one **Open(LINK)** outstanding. However, 802.2 and X.25 allow the possibility of multiple connections being handled through a single SNALink. In these cases, for each configured connection that is primed to await incoming calls, the local node will send an **Open(LINK)** with a blank connection name to the SNALink.

When an incoming call is received by the SNALink, the received XID should be passed to the local node on any LPI connection that is primed for incoming calls. The LPI connection selected must then be used for all future messages relating to that incoming call.

It is not necessary for the SNALink to perform validation of incoming calls—this will be performed by the local node. However, if required, the SNALink can choose to validate calls before passing them to the node. A common example of this is to ensure that only X.25 calls with a specific local address are passed through to the local node.

The following figure shows incoming call support with an SNALink that supports two connections. A remote station calls in and uses connection A. The node sets up connection B for incoming calls but then needs to open connection C. Because the SNALink only supports two connections, connection B is closed.

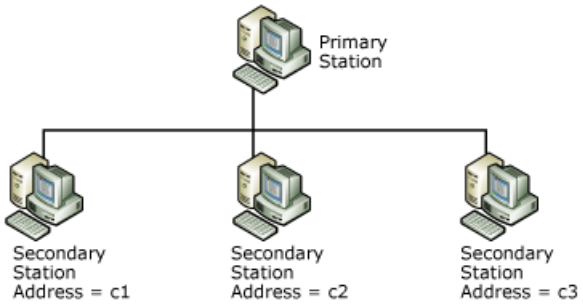
Call support with an SNALink that supports two connections



SDLC Multipoint Connections

The node can support primary multipoint (also known as multi-dropped) links, at both the primary and secondary end. Multipoint is a special configuration for a synchronous data link control (SDLC) leased link where a single SDLC line at the primary station can be used to communicate with up to 16 secondary stations. Special hardware is required to fan out the primary line so that there is a physical connection to each secondary station. The following figure shows an example with three secondary stations.

Primary station with three secondary stations



The SDLC address of the secondary station is used to route frames to and from the individual secondary stations. Hence, the SNALink at the secondary station needs to check the SDLC address as the primary sends all frames to all secondary stations. The SNALink at the secondary station should only accept frames with its SDLC address—the other frames should be ignored.

From the viewpoint of the node at a secondary station, the message flow at the data link control (DLC) interface is as for a point-to-point connection (described in Opening a Connection). The node need have no knowledge that this is a multipoint connection.

The primary end has to handle the special processing required for multipoint connections. The remainder of this section concentrates on the primary station.

At the primary end, there are the following LPI connections:

- One LINK LPI connection.
- A STATION LPI connection for each active secondary station.

Because the exchange identification (XID) exchange is carried out using the single LINK LPI connection, the Request-Open-Station and Send-XID messages always specify the station address of the secondary station that the XID has arrived from or is going to. Note that no XID is supplied on the Open(LINK) Request.

Each STATION LPI connection has different values of I, the index. After the station has been activated, data messages flow on the STATION LPI connection rather than the LINK LPI connection.

If the XID exchange fails because the secondary is failing to reply to the XID, the SNALink generates a special variant of link Outage message. Ideally, the SNALink would give a station Outage message, but this is not possible because the STATION LPI connection is not yet open. Instead, the SNALink generates a link Outage message with code 0xA0 and a subqualifier that is the SDLC address of the station.

When the stations are activated on a multipoint link, the majority of messages flow across the STATION LPI connections. If a connection to a particular secondary station is to be closed (because the operator deactivates it, for instance), the node issues a Close(STATION) Request. The SNALink replies with a Close(STATION) Response to the node and sends a Discontact (DISC) frame to the secondary station.

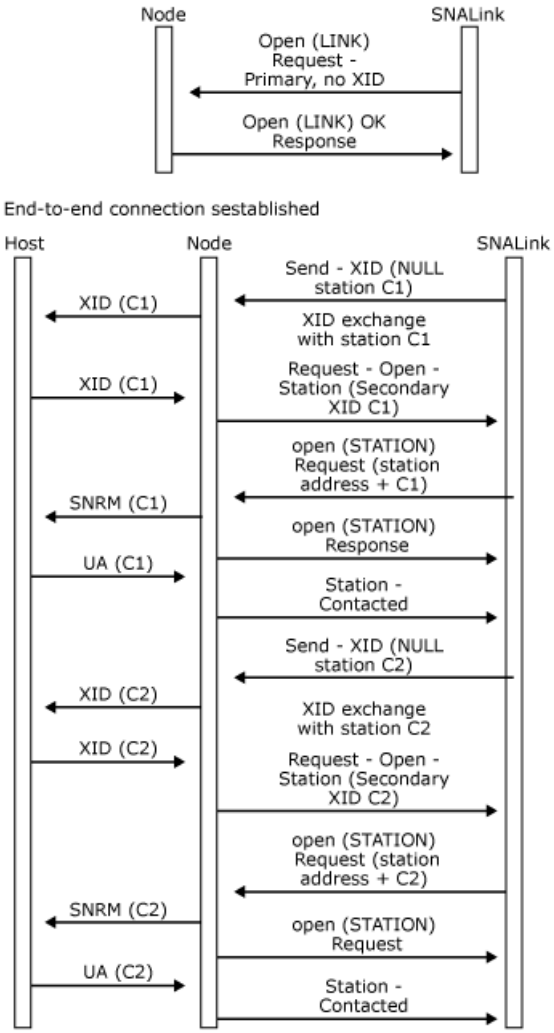
The SNALink can generate both station and link Outage messages. If the problem only affects a particular station, such as not responding to polls, the link generates a station Outage message and the node closes the station with a **Close(STATION) Request**. The SNALink responds with a **Close(STATION) Response**.

If the problem affects the link as a whole, such as the line being disconnected from the primary SDLC adapter, the SNALink generates a link Outage message and the node sends a Close(LINK) Request. The SNALink responds with a Close(LINK) Response.

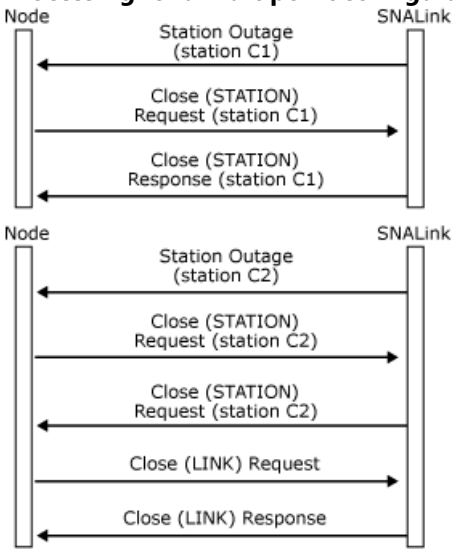
Whenever the node receives a **Close(STATION) Response**, it checks to see if any stations are still active on the multipoint link. If not, a **Close(LINK) Request** is sent. The SNALink responds with a **Close(LINK) Response**. The following figure shows the message flows for outage processing. It shows a multipoint connection with two secondary stations (the full XID exchange is

not shown).

Message flow for outage processing



Processing for a multipoint configuration with two secondary stations



Note that the station messages are labeled in the figure with station addresses. In fact, the node and SNALink use the LPI addresses to identify the two stations.

Setup Information

This section describes the integrated link service installation provided with Host Integration Server 2009.

In Host Integration Server, SNA Manager is used to install and configure link services. Host Integration Server uses the Microsoft System Installer (MSI) and MSI packages for the installation of the Host Integration Server software. Link services from independent hardware vendors (IHVs) are not included in the main Host Integration Server MSI packages. IHV link services are installed using a separate IHV-provided MSI package.

A sample IHV link service using the generic Synchronous Data Link Control (SDLC) link service that illustrates IHV link service installation is included in the Host Integration Server software development kit (SDK). The SDK samples are installed on your computer when the SDK option is selected during installation of Host Integration Server software. These sample files are also located on the Host Integration Server CD under the SDK\Samples\IHVLInks subdirectory. Host Integration Server does not support the earlier .inf-based link service setup procedure.

In This Section

Setup Registry Architecture

There are two main subtrees in the Microsoft Windows® 2000 registry where information is kept relevant to Host Integration Server 2009: the **SOFTWARE** tree and the **SYSTEM** tree. Both of these are subtrees of **HKEY_LOCAL_MACHINE**. The **SOFTWARE** tree contains generic information about independent hardware vendor (IHV) link services, and the **SYSTEM** tree contains information about the individual components of those services. While reading the following topics, it may be helpful to view examples of what is being discussed by inspecting the registry of an existing system with several of the built-in link services installed.

In This Section

Product Entries

All of the information relevant to the product as a whole resides in the registry under the key **SOFTWARE\Microsoft**. Each product or link support has an entry whose name consists of the product name and version separated by an underscore. This key contains most of the information about the product, such as the script name and option name that control it and the service name for that particular instance.

Each instance key must also have a NetRules key. This key contains all of the information for the Network Control Panel Applet bindings.

The Host Integration Server Setup writes the path of the root directory of the computer's Host Integration Server tree into the key:

SOFTWARE\Microsoft\SNA Server\CurrentVersion\Setup\RootDir

Service Entries

Each instance of a component appears to the system as a unique service. These services must be created using the Service Control Manager (SCM). The SCM creates a registry entry for each service under **SYSTEM\CurrentControlSet\Services**. This key contains all of the service-specific information.

The top-level service key contains information that the SCM uses to control the service. This includes the type of service that this key represents, how it should be started, what sort of error handling should be used, the path to the executable image, and so on. All information in this key should be handled by the SCM. Each service key also contains two subkeys—the Linkage key and the Parameters key.

The Linkage key is used by the Network Control Panel Applet to store binding information. The Parameters key contains information that is relevant to Host Integration Server 2009 Setup, such as the name of the DLL responsible for handling a link service. All information in this key should be handled by Host Integration Server 2009 Server Setup. The Parameters key contains another key, ExtraParameters, which is used for any IHV-specific information, including component-specific parameters and other information not required by the main SNA Server Setup program.

Integrated Link Service Setup on Host Integration Server

In Host Integration Server 2009, the SNA Manager supports installation and configuration of link services. Host Integration Server uses the Microsoft System Installer (MSI) and MSI packages for the installation of the Host Integration Server software. Link services from independent hardware vendors (IHVs) are not included in the main Host Integration Server MSI packages. IHV link services are installed using a separate IHV-provided MSI package. For an example of this process, install the DLC, 802.2, or IBM Synchronous Data Link Control (SDLC) link service in Host Integration Server.

IHV MSI Packages contain two types of features:

- Features that can be installed and used independently of Host Integration Server.
- Features that require Host Integration Server to function.

Features that can be installed and used independently of Host Integration Server include drivers, utilities, and applications that can run without Host Integration Server support. These features should be represented in the package as one or more features in the MSI Select Features dialog box. (For more information, see the Generic Link Service sample feature "Generic Link Service" from the Generic.msi Featuretable.) Generic.msi can be found in the Host Integration Server SDK on the CD and in the install directory at the following address: SDK\Samples\IHVLinks\Packages\Generic.msi.

Features that require Host Integration Server include drivers, utilities, and applications that require Host Integration Server to function. These features should be represented in the package as one or more features in the MSI Select Features dialog box. These features should be hidden if Host Integration Server is not installed on the computer. (For more information, see the Generic Link Service sample feature "Host Integration Server Support" from the Generic.msi feature table.)

Properties can be equated to a variable (either global or local) in a high-level programming language such as C or C++. Properties can be used as a placeholder for informational text, or as values used during an installation. (For more information, see the property SERVER_INSTALLED in the custom action source code GenSet.cpp, and the Condition table entry in the Generic.msi package.)

Custom Actions provide a method of extending the capabilities of MSI. Functions not supported in MSI, can be custom written, and invoked from within a sequence table or directly from a dialog control event. (For more information, see the Custom Actions SetHISPath and GetHISData in the GenSet.cpp source file.)

Launch Conditions provide a method of preventing an install from launching. The sample MSI package included with the Host Integration Server SDK does not use a launch condition, however, if your package requires Host Integration Server to be installed for your features to function, you should include a launch condition that fails the installation if Host Integration Server is not detected.

The Condition table provides a method of controlling the layout of the features listed in the select feature dialog box. The sample MSI package uses the Condition table to hide the "Host Integration Server Support" if Host Integration Server is not detected in the installation.

For Host Integration Server to control the installed state of IHV features that require Host Integration Server to be installed, the following registry keys must be included in the package for each separate feature which requires Host Integration Server to be installed.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SnaBase\IHVSupport\<ihv key>
<feature table entry 1> : REG_SZ : <product code 1>
<feature table entry 2> : REG_SZ : <product code 2>
.
.
.
<feature table entry n> : REG_SZ : <product code n>
```

Where:

<feature table entry x> specifies an entry in the feature table (not title).

<product code x> specifies the package product code.

Note that feature table entries must be unique. Product codes may or may not be unique depending on the number of packages provided by an IHV. One package may have several Host Integration Server dependent features and therefore should list each feature separately.

These registry keys should be installed by the feature rather than globally by the package.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SnaBase\LinkServicesInstalled
<Link Service Name> : REG_SZ : <Link Service Configuration Dll name>
```

Target paths specify a directory where files will be installed. An MSI package will contain one or more target paths.

The sample package contains two features:

- Generic Link Service
- Host Integration Server Support

The Generic Link Service is not dependent on Host Integration Server. This feature installs the following:

- gencfg.exe into the <Generic Link Service> directory.
- generic.sys driver into the %windir%\system32\drivers directory.
- generic.inf into the %windir%\inf directory

Host Integration Server Support is dependent on Host Integration Server. This feature installs the following:

- gendtct.dll into the HIS\system directory.
- generic.dll into the HIS\system\hwsetup\i386 directory.

This feature contains the following entry in the condition table to hide the feature if Host Integration Server is not detected

```
HIS_RELATED_FEATURE  0  SERVER_INSTALLED="NO"
```

See dialog snapshots later in this topic for layout of features with and without Host Integration Server installed:

Adds:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SnaBase\LinkServicesInstalled
Generic Link Service : REG_SZ : GENERIC.DLL

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SnaBase\IHVSupport\GenericLinkService
HIS_RELATED_FEATURE : REG_SZ : {FDF11E0E-3BFF-4B0F-89BD-E4E1FB979E4D}
```

The sample package uses one custom action DLL with two entry points:

```
SetHISPath
GetHISData
```

The **SetHISPath** entry sets the target path to the Host Integration Server installation directory.

The **GetHISData** entry sets the MSI property SERVER_INSTALLED to "YES" if Host Integration Server is installed and sets the MSI property SERVER_INSTALLED to "NO" if Host Integration Server is not installed.

The sample contains two target paths:

```
INSTALLDIR
INSTALLDIR1
```

The INSTALLDIR target path specifies the installation directory for the features that are not dependent on Host Integration Server. It can be set using the **browse** button in the **Select Features** dialog box when the Generic Link Service feature is currently selected.

The INSTALLDIR1 target path specifies the directory where Host Integration Server is installed. This target path is set by the custom action SetHISPath.

The IHVUtil.exe tool can be used to verify the Host Integration Server dependent interfaces. If the tool is launched after the IHV package is installed, all Host Integration Server dependent features should show up in the dialog box. Executing Remove from the dialog box should remove the Host Integration Server dependent feature as well as remove it from the dialog box.

Note that the sample provided can also be tested using the SNA Manager. While the sample does not actually function, it will appear as an installed link service.

In This Section

Integrated Link Service Configuration and Reconfiguration on Host Integration Server

Constructing an Integrated Link Service DLL on Host Integration Server

Integrated Link Service Configuration and Reconfiguration on Host Integration Server

In Host Integration Server 2009, the initial configuration functions are performed by your configuration DLL running in the context of SNA Manager.

After your link service has been created, SNA Manager must be able to locate its configuration DLL when the operator wishes to reconfigure the link service. To support this feature, when your configuration DLL initially creates the link service, it must put a new value in the registry of the target server as follows:

SYSTEM\CurrentControlSet\Services\<yourLinkService>\Parameters

DLLName: REG_SZ: <configDllName>

where:

<*configDllName*> is the file name and extension of the configuration DLL, for example, IBMSDCFG.DLL. No path is specified in the value.

This value replaces **InfName**, which was used in SNA Server 2.*x* to name the path to the .inf file.

Since SNA Manager can be running on a management workstation remote from the target server, the configuration DLL must be able to create configuration information on the target server. Host Integration Server loads the appropriate configuration DLL over the network from \<snaRoot>\SYSTEM\HWSETUP\<cpu> on the target server as needed.

Note
There is an alternate way of locating the link service configuration DLL (linkcfg) if the link services from the vendor were not i ncluded with the released Host Integration Server CD.

Note
Depending on the setup tool used by the vendor, the vendor's setup software may not be able to read the registry and locate the directory where link services should be installed. To resolve this problem, the SNA Manager scans the **LinkServicesInsta lled** key prior to making the call to the link service configuration DLL. The SNA Manager checks for a % character in the *confi gDllName* and if it exists, *configDllName* will be interpreted differently than just the name of the configuration DLL. The follo wing example illustrates this case:

Note
Under the **SYSTEM\CurrentControlSet\Services\<yourLinkService>\Parameters key**

Note
DLLName: REG_SZ: "share\%s\<relative path and DLL Name>

Note
If a %s string is found, \\ServerName will be prepended and the CPU architecture string (i386) will be substituted for %s.

Constructing an Integrated Link Service DLL on Host Integration Server

Microsoft Host Integration Server 2009 provides an enhanced method for installing integrated link services that allows for remote setup and administration of new link services, as well as support for setup and configuration using a command-line tool. This feature is based on the link service provider supplying a setup and configuration DLL exporting a specific list of functions. A developer must follow certain standards for using this SNA link service configuration DLL (linkcfg) and set various keys and values as registry settings to be used by Host Integration Server for link service configuration.

To support vendors using these setup and configuration features, Host Integration Server includes the source code for a sample generic Synchronous Data Link Control (SDLC) link service configuration DLL. Also included for use by developers is the source code to a library of utility functions (lnktools) that are commonly useful when implementing the linkcfg DLL. This sample code and the documentation that follows can be used as a guideline for vendors developing similar link service configuration DLLs for their hardware. This sample source code is located on the Host Integration Server CD in the \SDK\SAMPLES\IHVLinks subdirectory. Sample include files, C++ files, resource files, makefiles, and project files are included for use with Microsoft Visual C++ version 6.0 and later.

In This Section

Components of an Integrated Link Service Configuration DLL on Host Integration Server

The link service configuration DLL (linkcfg) must export the following functions.

Exported function	Purpose
CommandLineAdd	Called from **LinkCfg** to parse command-line input.
ConfigureLinkService	Called from SNA Manager to add or modify a link service.
ConfigureLinkServiceEx	Called from SNA Manager to add or modify a link service, returning a configuration buffer to be added to the configuration file.
DisplayHelpInfo	Returns a buffer containing command-line syntax for this type of link service.
RemoveLinkService	Called from SNA Manager to remove a link service.
RemoveAllLinkServices	Called from Setup to remove all instances of this link service.

The sample linkcfg.cpp DLL is written in C++ using the Microsoft Foundation Classes (MFC) and uses a single property sheet with two property pages as follows:

- The card configuration property page implementation is in the cardcfg.cpp and cardcfg.h files. This property page is concerned with configuring various hardware properties (interrupt, DMA channel, and I/O address, for example) of the link service hardware.

- The connection mode property sheet implementation is in the mode.cpp and mode.h files. This property page is concerned with configuring mode information (link service name, link service title, Synchronous Data Link Control (SDLC) line type, for example) for the link service.

The two property pages are linked to the link service property sheet in linkcfg.cpp within the **ConfigureLS** routine. This function is called by the exported **ConfigureLinkService** and **ConfigureLinkServiceEx** routines in linkcfg.cpp. An actual link service configuration DLL developed from these sources may require more property pages depending on the information needed to configure the actual link service DLL.

The registry.h include file used by linkcfg.cpp contains a global definition of the registry entries required for the sample generic SDLC link service. The values in this structure will be modified to contain the actual information specified by the user. This structure is added to the registry when a new link service is configured, and this structure is removed when a link service is deleted. The registry values that a developer must modify include the Link Registry Base entry (LINKSERVICE is used in the sample include file), the name of the device driver root (GenSdlc is used in the sample include file and source code), and various software and service registry settings appropriate for the target link service.

Several of the exported link service DLL functions use a configuration buffer, the CONFIG_BUFFER structure defined in linkcfg.h. The format of any CONFIG_BUFFER used by developers must match the structure format of this sample file for the first three parameters. Other parameters may differ for a developer's version of the CONFIG_BUFFER structure based on the target link service.

The sample link service configuration DLL calls a set of general utility functions that are not specific to any target link service. These utility functions are included in the lnktools library (lnktool.cpp) that is linked in as an OBJ file. This lnktools library includes the following utility functions that are useful in developing link service configuration DLLs.

Utility function	Purpose
AddPerfmonCounters	Add Perfmon counters for this link service.
bCreateService	Create a service on a computer.
bDeleteService	Delete a service on a computer.
bStopService	Stop a service running on a computer.

CheckForExistingLinkService	Check to see if a link service of this type exists with this title.
ConvertHexStringToDWORD	Convert a hexadecimal string to a DWORD value.
ExtractNextParameter	Get the next parameter from a buffer.
fAddRegistryEntry	Add a new registry value to the registry.
fCanWeAdministerRemoteBox	Determine if the user has administrative privileges on the remote computer.
fConnectRegistry	Connect to a remote computer's registry and return a handle to the remote registry.
fDisconnectRegistry	Disconnect from a remote computer's registry.
fFindAndReplaceString	Find and replace a substring within a string.
fFindString	Determine if a string exists within a string buffer.
fFindStringInMultiSZ	Find a string in a REG_MULTI_SZ string list and return entire string.
fQueryRegistryValue	Query a value from the registry.
fRegistryKeyExists	Test whether a registry key exists.
fRemoveRegistryEntry	Remove a registry key.
fRemoveRegistryValue	Remove a registry value.
fStringCompare	Determine if two strings compare.
LoadStringResource	Load a string from the string resource.
ParseNextField	Return the next field from a string.
RemovePerfmonCounters	Remove Perfmon counters for this link service.
ReturnString	Return a pointer to a string resource string.

The sample source code for a generic SDLC link service configuration DLL (linkcfg) includes several functions that may be useful as sample code when developing link service configuration DLLs for other hardware. The following functions are included in the linkcfg.cpp source code that may be of use as examples.

Utility function	Purpose
bDetectNetworkCard	Detect the remote network card and return the card settings buffer for the sample generic SDLC link service.
bLastGenericDFTLinkService	Check for the last generic SDLC link service for the sample generic SDLC link service. This routine is used to determine if the GENSDLC Device Driver (if one exists) can be removed.
ConfigureLS	The common link service configuration function used by the sample generic SDLC link service.
fAddAllRegistryValues	Add all registry values for the sample generic SDLC link service.
fAddClassAndBindformRegistries	Add the class and bindform registry entries for the sample generic SDLC link service. The bindform and class registry entries can only exist for the first link service of this type.
fEnumerateEventLogSources	Enumerate the Event Log sources registry value for the sample generic SDLC link service.
fRemoveAllRegistryValues	Remove all registry values for the sample generic SDLC link service.
fReplaceAllRegistryValues	Replace all user-provided information in the registry data for the sample generic SDLC link service.

fReplaceRegistryData	Replace global registry data for the sample generic SDLC link service.
fReplaceRegistryKeyName	Replace global registry structure strings for the registry key name for the sample generic SDLC link service.
fSetupGlobalValues	Create or update all user-provided information in the registry data structure for the sample generic SDLC link service.
InitializeGlobalStructure	Initialize link service data contained in the global data structure for the sample generic SDLC link service.

Contents of IHVLinks Sample Kit on Host Integration Server

The sample source code for a generic Synchronous Data Link Control (SDLC) integrated link configuration MSI package that illustrates an integrated independent hardware vendor (IHV) link service installation are included on the Host Integration Server 2009 CD. These sample programs are located in the \SDK\Samples\IHVLinks subdirectory on the Host Integration Server CD. These files are copied to your hard drive during Host Integration Server software or Host Integration Client software installation when the Host Integration Server Software Development Kit (SDK) option is selected. These samples are installed in the SDK\Samples\IHVLinks subdirectory below where the Host Integration Server SDK software is installed (C:\Program Files\Microsoft Host Integration Server\SDK, by default).

These sample files include the following directories.

Directory	Description
LinkServ	Directories used in creating the IHVLinks sample generic SDLC link service configuration and detection DLLs.
LinkServ\Build	A file containing the normal COFF base address for a link service configuration DLL (linkcfg).
LinkServ\Detect	The source code to the sample generic SDLC link service detection DLL.
LinkServ\Linkcfg	The source code to the sample generic SDLC link service configuration DLL. The link service configuration DLL must export specific functions. A definitions (.DEF) file must be used so that exported function names are not decorated by the compiler and linker.
LinkServ\LnkTools	The source code to a collection of library routines used by the generic SDLC link service configuration DLL.
LinkServ\NT5INF	An INF file that can be used with the generic SDLC link service.
Setup	Directories used in creating the IHVLinks sample generic SDLC link service setup and setup test tools.
Setup\Bins	The compiled generic MSI package containing the generic SDLC link service driver, configuration and detection DLLs, and configuration tool.
Setup\CASource	This directory contains the source code used for the custom actions in setup. The GetHISData custom action sets the MSI property SERVER_INSTALLED to "YES" if Host Integration Server is installed, otherwise the property is set to "NO". This custom action is used to disable Host Integration Server dependent features from the installation directory if not installed. The SetHISPath custom action sets the target directory INSTALLDIR1 to the installation directory where Host Integration Server was installed. This custom action is used to set the destination directory for the link service configuration DLLs.
Setup\Package	The sample GENERIC.MSI SDLC link service package ready for installation and testing.

Setup\Tools	The source code for various utility functions and tools that can be used to test your integrated link service MSI package.

This sample source code is included for your reference. The communication between workstation management station and Host Integration Server is performed by RPCSVC.EXE, the SNA remote procedure call (RPC) service.

Compiling and Linking a SNALink

This section provides information about compiling and linking a SNALink for use with Host Integration Server 2009. This section also lists and explains the header files and libraries need to build a SNALink.

In This Section

Host Integration Server DLC Header Files

The following files are required to build a Host Integration Server 2009 SNALink:

File	Description
SNA_DLC.H	Main header file containing the definitions of buffer and message formats.
SNA_CNST.H	Function prototypes for the Base/DMOD interface calls and constant definitions.
TRACE.H	Definitions of the logging and tracing macros.
IHVLINK.LIB	Main library for the SNADIS interface.

Included Files

To compile a SNALink, the header files SNA_DLC.H, SNA_CNST.H and TRACE.H are required. In addition, one of the standard operating system header files may be required. To include the required files, the following lines should be used in your application:

```
#include <sna_dlc.h>
#include <sna_cnst.h>
#include <trace.h>
```

Note

The TRACE.H include file is required to enable SNA tracing and use of the Host Integration Server 2009 Trace viewer utility.

Required Exports

The IHV link support DLL must export the following entry points:

- SNALinkInitialize
- SNALinkWorkProc
- SNALinkDispatchProc

These are called by the Base scheduler when the SNALink is invoked.

Compiler Options

When compiling the SNALink DLL, the following compiler options are required:

Option	Explanation
/c	Compile only, without linking. Linking is done as a separate phase to include the required Microsoft Host Integration Server 2009 libraries.
/D NOTRC	The NOTRC macro specifies that internal tracing should not be compiled into the application. The /D NOTRC option should be used for building a final system (internal tracing should not be included because it will degrade performance and occupancy). For a development system, you may want to compile with internal tracing; if so, remove the /D NOTRC option.
/D WIN32_SUPPORT	The macro WIN32_SUPPORT is used in the header files SNA_DLC.H, SNA_CNST.H, and TRACE.H to support variants of the DLC interface for Microsoft Windows 2000.
/Gzs	**z**: Use stdcall conventions .**s**: Remove stack check calls.

The following compiler flags are required, but any of the valid options for each flag may be used, as appropriate to your application:

/O	Optimization
/W	Warning level

Linking

The IHVLINK.LIB library must be linked with the application. It contains the Base, DMOD imports, and diagnostics routines. The DMOD and the Base are implemented as DLLs.

Note
The Host Integration Server 2009 library only contains the external references for the corresponding DLLs, which are part of t he main Host Integration Server product.

Synchronous Dumb Card Interface

This section describes the interface to the synchronous dumb card device driver used by the Synchronous Data Link Control (SDLC) and X.25 SNALinks that Microsoft Host Integration Server 2009 supplies. The interface provides a simple but flexible mechanism for transferring frames of data through a dumb synchronous communications card (such as the IBM MPCA card).

This interface is intended primarily for independent hardware vendors (IHVs) who want to provide drivers for their own dumb cards that are directly compatible with SDLC and X.25 SNALinks. It can also be used by Independent Hardware Vendors (IHVs) who intend to write their own SNALinks and want to ensure that their drivers conform to the standard Host Integration Server 2009 model.

In This Section

Driver Interface

Application software running on Microsoft Windows Server 2003 or Windows 2000 normally does not interact directly with device drivers. Usually, the operating system itself controls the interface to underlying device drivers on behalf of the application. For example, Disk I/O consists of sequences of driver requests generated by a file system, as a result of an application making file system requests.

By contrast, in the Microsoft Host Integration Server 2009 device driver model, synchronous dumb cards are controlled directly by the SNALink using input and output control (IOCTL) commands. This mechanism enables the SNALink to pass raw control packets to the driver without any intervention from the operating system.

This is achieved by issuing an Open request with a file name that identifies the device driver. The operating system detects the fact that this file is a driver and passes an OPEN I/O request packet to the driver. The user application is returned a handle that can be used to reference the driver.

The IBMSYNC driver creates various device names. During setup, the configuration for adapters in the computer is saved in the registry. When the driver starts, it reads this data and creates the device names for all the adapters that are found.

The following table lists the device names that the IBMSYNC driver can create.

Device name	**Description**
\Device\IBMSDLC	Standard IBM Synchronous Data Link Control (SDLC) adapter.
\Device\MPCA_1	IBM MPCA 1 adapter. This adapter has a switch set on it to enforce MPCA 1 operation. This adapter is the primary MPCA adapter in the computer and supports direct memory access (DMA) interrupt mode.
\Device\MPCA_2	IBM MPCA 2 adapter. This adapter has a switch set on it to enforce MPCA 2 operation. This adapter is the secondary MPCA adapter in the computer and supports only interrupt mode.
\Device\SYNC_*x*	Generic adapter (for example, Microgate). The letter *x* is 1 (for the primary adapter) or 2 (for the secondary adapter).
\Device\MPAA_S*x*	IBM MPAA adapter, where *x* represents the number of the MCA slot where the adapter is installed in the computer. This number is a value from 1 through 8.
\Device\SYNC_S*x*	Generic MPAA adapter (for example, the Microgate MPAA adapter). The letter *x* represents the number of the MCA slot where the adapter is installed in the computer. This number is a value from 1 through 8.

Subsequent IOCTL calls using **DeviceIOControl** made by the SNALink using the driver handle cause the operating system to pass an IOCTL I/O request packet to the driver. The driver therefore sees IOCTL requests from the SNALink as a series of I/O request packets passed to it by the operating system.

The Host Integration Server 2009 dumb card interface uses the following operating system calls:

- **OpenFile**
- **DeviceIOControl**
- **CloseFile**

DeviceIOControl allows free-format information to be passed to the driver. The dumb card interface uses its own format of information to pass all requests to the driver (with the exception of Open and Close requests, which are handled differently by the operating system).

In This Section

- Architecture Overview

Architecture Overview

This section describes how information is transferred between the driver and the SNALink.

In This Section

- Interface Record
- Event Signaling
- Link Characteristics

Interface Record

Status information is transferred between the driver and the SNALink using a buffer known as the interface record.

The driver allocates this buffer when it starts and maintains the information in it while it is running. The contents of this buffer are copied to an SNALink buffer by using an IOCTL call of type **READ_INTERFACE_RECORD**.

Event Signaling

The device driver notifies the SNALink whenever an event occurs (such as a frame being received from the line) by setting an event.

The SNALink provides the driver with a handle to this event (or semaphore) at the start of day by issuing an IOCTL call of type **SET_EVENT_HANDLE**.

Link Characteristics

Before the driver can transfer any data, it needs information about the link. This includes the following:

- The frame size.
- The station address to listen on (if required).
- Details of hardware selectable options, such as SDLC/HDLC, Internal/External clocking, and so on.

For more details of these options, refer to the topic Function 0x42: Set Link Characteristics.

I/O Request Packets

All I/O requests are passed to the driver by Microsoft® Windows Server™ 2003 or Windows® 2000 using the standard IRP structure. For more details about this, refer to the Windows Server 2003 or Windows 2000 Device Driver Kit.

I/O request packets are defined in terms of C structures. The relevant fields are accessed as follows:

Field name	Description
IRP.CurrentStackLocation -> MajorFunction	Defines the IRP as an IOCTL.
IRP.IoStatus	Status codes upon completion of request.
IRP.CurrentStackLocation -> IoControlCode	The IOCTL function code.

IoControlCode identifies the function to be performed and **IoStatus** is the mechanism for returning result codes to the SNALink. The structure **IoStatus** is defined as follows:

IoStatus.Status

A standard Windows Server 2003 or Windows 2000 result code (for example, STATUS_INVALID_PARAMETER) as defined in the Windows Server 2003 or Windows 2000 header file NTSTATUS.H.

IoStatus.Information

For successful read-frame IOCTLs, the length of the received buffer (can be zero if no data available). Additional error information, as defined in the header file SECLINK.H.

In This Section

Initialization

Device drivers under Windows Server 2003 or Windows 2000 should perform all initialization required at the start of day when they are loaded by Windows Server 2003 or Windows 2000. Configuration information for device drivers is stored in the Configuration Registry under Windows Server 2003 or Windows 2000. For more details, refer to the documentation supplied with the Windows Server 2003 or Windows 2000 Device Driver Kit.

The SNALinks for dumb cards are implemented by the following files:

SDLC SNALink
SNAroot\SYSTEM\IBMSDLC.DLL

X.25 SNALink
SNAroot\SYSTEM\IBMX25.DLL

To bind to one of these, an installation script should mention the appropriate DLL in the IHVDLL registry entry that the script creates for the link service.

OPEN Call

The **OPEN** call has no parameters. It grants access to the driver from a particular process. The driver ensures that only one **OPEN** is accepted by the link at any one time. When **OPEN** is processed, the driver attempts to reserve access to hardware resources such as interrupt vectors. The **OPEN** is rejected if this fails.

After a successful **OPEN** request, the driver expects to receive the following IOCTL commands:

- **SET_EVENT_HANDLE**
- **SET_INTERFACE_RECORD**
- **SET_LINK_CHARACTERISTICS**

Of these, the first two can be performed in any order, but both should be issued before calling **SET_LINK_CHARACTERISTICS**.

When these three calls have been successfully performed by the SNALink, the driver is ready for information transfer.

CLOSE Call

The **CLOSE** call has no parameters. It performs the logical converse of **OPEN**. Resources are released back to the operating system when a **CLOSE** is performed.

IOCTL Command Summary

The parameters to the IOCTL request packet are stored in the following fields in the associated I/O request packet (IRP).

IRP.CurrentStackLocation -> IOControlCode	**Function code**
IRP.SystemBuffer	Address of parameter buffer (if used)
IRP.CurrentStackLocation -> InputBufferLength	Length of parameter buffer
IRP.UserBuffer	Address of data buffer
IRP.CurrentStackLocation -> OutputBufferLength	Length of data buffer

Note that under Windows Server 2003 or Windows 2000, the operating system reserves the lower four bits of the IOCTL function codes to determine the method used to map the various buffers passed on the **DeviceIoControl** function call into the driver address space. The various options available to device driver writers are:

Low nibble	**IOCTL definition**
0	METHOD_BUFFERED
1	METHOD_IN_DIRECT
2	METHOD_OUT_DIRECT
3	METHOD_NEITHER

For further details of the memory mapping performed by these various options, refer to the Windows Server 2003, Windows 2000, or Windows DDK documentation.

For a driver function code of ZZ, using memory mapping code M, the IOCTL code passed on the **DeviceIoControl** function call is 0xZZM.

The function codes are set out as shown in the table later in this topic. Note that all other function codes will be returned with the error ERROR_INVALID_DEVICE_REQUEST in the field **IoStatus.Status**. The Windows Server 2003 or Windows 2000 I/O System validates the address and length of the areas passed as parameter and data packets. If the address validation fails, an exception will be raised.

All requests must return immediately. In general, they are simple, immediate operations, but in the case of Transmit Frame and Receive Frame, the driver must not suspend the calling SNALink thread while waiting for I/O to complete—a relevant return code should be sent instead, allowing the SNALink to retry.

The complete list of functions is as follows.

Function	**Function code**	**Windows Server 2003 or Windows 2000 IOCTL code**
Function 0x41: Set Event/Semaphore Handle	0x41	0x410
Function 0x42: Set Link Characteristics	0x42	0x420
Function 0x43: Set V24 Output Status	0x43	0x430
Function 0x44: Transmit Frame	0x44	0x441
Function 0x45: Abort Transmitter	0x45	0x450
Function 0x46: Abort Receiver	0x46	0x460
Function 0x47: Off-Board Load	0x47	0x470
Function 0x61: Get/Set Interface Record	0x61	0x613
Function 0x62: Get V24 Status	0x62	0x622
Function 0x63: Receive Frame	0x63	0x632

Function 0x64: Read Interface Record	0x64	0x642

In the function descriptions in the following topics, the bit-numbering convention is: The bits in a byte are numbered 0 through 7, where bit 0 is the least significant and bit 7 is the most significant.

Note

There is no function for the controlling autodialer across the synchronous dumb card interface. This autodial feature is implemented in the link service itself. The Microsoft link services that support the synchronous dumb card interface first perform the dial operation by sending the dial string containing the server-stored number to a COM port rather than the SDLC chip, and then sending a command to the device driver to raise DTR through a Set V24 Output Status IOCTL.

Equates and Structure Layouts

Many standard operating system device driver error codes are used (for example, "invalid function"), together with a new set of device driver-specific errors. Return codes below 0x80 reflect serious failures.

```
/* Copyright Data Connection Ltd. 1989 */_
/*****************************************************************************/
/* Link Device Driver interface constants and structures. */
/*****************************************************************************/
/*****************************************************************************/
/* WIN32  16/04/92  SW   Added more helpful names from WIN32 hdr file        */
/* IHV    03/06/92  MF2  Add semfisui.h                                      */
/*****************************************************************************/

/*****************************************************************************/
/* This include file is used in 5 subsystems                                 */
/*                                                                           */
/* - the NT   driver                          LINK_NTDRIVER     */
/* - the X25  link service for NT             LINK_NTX25        */
/* - the SDLC link service for NT             LINK_NTSDLC       */
/* - the X25  link service for OS/2           LINK_OS2X25       */
/* - the SDLC link service for OS/2           LINK_OS2SDLC      */
/*                                                                           */
/* (The OS/2 driver doesn't count because it is in assembler). */
/*                                                                           */
/* These are distinguished by #defines as set in the following */
/*                                                                           */
/*****************************************************************************/

#ifdef        IMADRIVER
  #define     LINK_NTDRIVER
#else
  #ifdef      SDLC
    #ifdef    WIN32
      #define LINK_NTSDLC
    #else
      #define LINK_OS2SDLC
    #endif
  #else
    #ifdef    WIN32
      #define LINK_NTX25
    #else
      #define LINK_OS2X25
    #endif
  #endif
#endif
/*****************************************************************************/
/* Device function codes for DosDevIOCtl to link device driver               */
/*****************************************************************************/
#ifdef WIN32               /* WIN32 constants defined in semfisui.h         */
#define IoctlCodeSetEvent               0x410
#define IoctlCodeSetLinkChar            0x420
#define IoctlCodeSetV24                 0x430
#define IoctlCodeTxFrame                0x440
#define IoctlCodeAbortTransmit          0x450
#define IoctlCodeAbortReceiver          0x460
#define IoctlCodeSetInterfaceRecord     0x610 /*IRMdl?*/
#define IoctlCodeGetV24                 0x623
#define IoctlCodeRxFrame                0x633
#define IoctlCodeReadInterfaceRecord    0x643   /*IRMdl?*/
#else
//obsolete names from previous version
//#define CELDDSSH 0x41       /* Set Semaphore Handle                          */
//#define CELDDSLC 0x42       /* Set Link Characteristics                      */
//#define CELDDSVS 0x43       /* Set V24 Output status                         */
//#define CELDDTXF 0x44       /* Transmit a frame of data                      */
```

```
//#define CELDDATX 0x45       /* Abort Transmitter                          */
//#define CELDDARX 0x46       /* Abort Receiver                             */
//#define CELDDGIR 0x61       /* Get Interface Record Address               */
//#define CELDDGVS 0x62       /* Get V24 Input Status                       */
//#define CELDDRXF 0x63       /* Receive a frame of data                    */
//#define CELDDCAT 0x82       /* Device function category code              */
//
// new names

#define IoctlCodeSetEvent              0x41
#define IoctlCodeSetLinkChar           0x42
#define IoctlCodeSetV24                0x43
#define IoctlCodeTxFrame               0x44
#define IoctlCodeAbortTransmit         0x45
#define IoctlCodeAbortReceiver         0x46
#define IoctlCodeSetInterfaceRecord    0x61
#define IoctlCodeGetV24                0x62
#define IoctlCodeRxFrame               0x63

#endif

/*************************************************************************/
/* Constants for the driver-specific IOCtl return codes.                 */
/*************************************************************************/
#define CEDNODMA 0xff80     /* Warning (NO DMA!) from set link chrctrstcs    */
/*************************************************************************/
/* Equates for the link options byte 1.                                  */
/*************************************************************************/
#define CEL4WIRE 0x80
#define CELNRZI  0x40
#define CELPDPLX 0x20
#define CELSDPLX 0x10
#define CELCLOCK 0x08
#define CELDSRS  0x04
#define CELSTNBY 0x02
#define CELDMA   0x01

/*************************************************************************/
/* Equates for the driver set link characteristics byte 1.               */
/*************************************************************************/
#define CED4WIRE 0x80
#define CEDNRZI  0x40
#define CEDHDLC  0x20
#define CEDFDPLX 0x10
#define CEDCLOCK 0x08
#define CEDDMA   0x04
#define CEDRSTAT 0x02
#define CEDCSTAT 0x01

/* Nicer names for NT-style code */

#define LinkOption_4Wire           CED4WIRE
#define LinkOption_NRZI            CEDNRZI
#define LinkOption_HDLC            CEDHDLC
#define LinkOption_FullDuplex      CEDFDPLX
#define LinkOption_InternalClock   CEDCLOCK
#define LinkOption_DMA             CEDDMA
#define LinkOption_ResetStatistics CEDRSTAT

/*************************************************************************/
/* Equates for the output V24 interface flags.                           */
/*************************************************************************/
#define CED24RTS 0x01
#define CED24DTR 0x02
#define CED24DRS 0x04
#define CED24SLS 0x08
#define CED24TST 0x10
```

```
/* Nicer names for NT-style code */

#define IR_OV24RTS  CED24RTS
#define IR_OV24DTR  CED24DTR
#define IR_OV24DSRS CED24DRS
#define IR_OV24SlSt CED24SLS
#define IR_OV24Test CED24TST


/*****************************************************************************/
/* Equates for the input V24 interface flags.                                */
/*****************************************************************************/
#define CED24CTS 0x01
#define CED24DSR 0x02
#define CED24DCD 0x04
#define CED24RI  0x08

/* Nicer names for NT-style code */

#define IR_IV24CTS  CED24CTS
#define IR_IV24DSR  CED24DSR
#define IR_IV24DCD  CED24DCD
#define IR_IV24RI   CED24RI
#define IR_IV24Test 0x10


/*****************************************************************************/
/* Structure for the device driver interface record.                         */
/*****************************************************************************/

#define CEDSTCRC  0         /* Frames received with incorrect CRC            */
#define CEDSTOFL  1         /* Frames received longer than the maximum       */
#define CEDSTUFL  2         /* Frames received less than 4 octets long       */
#define CEDSTSPR  3         /* Frames received ending on a non-octet bndry   */
#define CEDSTABT  4         /* Aborted frames received                       */
#define CEDSTTXU  5         /* Transmitter interrupt underruns               */
#define CEDSTRXO  6         /* Receiver    interrupt overruns                */
#define CEDSTDCD  7         /* DCD (RLSD) lost during frame reception        */
#define CEDSTCTS  8         /* CTS lost while transmitting                   */
#define CEDSTDSR  9         /* DSR drops                                     */
#define CEDSTHDW 10         /* Hardware failures - adapter errors            */

#define CEDSTMAX 11

#define SA_CRC_Error       CEDSTCRC
#define SA_RxFrameTooBig   CEDSTOFL
#define SA_RxFrameTooShort CEDSTUFL
#define SA_Spare           CEDSTSPR
#define SA_RxAbort         CEDSTABT
#define SA_TxUnderrun      CEDSTTXU
#define SA_RxOverrun       CEDSTRXO
#define SA_DCDDrop         CEDSTDCD
#define SA_CTSDrop         CEDSTCTS
#define SA_DSRDrop         CEDSTDSR
#define SA_HardwareError   CEDSTHDW  /* e.g. CmdBufferFull not set  */

#define SA_Max_Stat        CEDSTMAX

#ifdef WIN32

typedef struct         _INTERFACE_RECORD
{
  int                   RxFrameCount;   /* incremented after each frame rx'd */
  int                   TxMaxFrSizeNow; /* max available frame size av. now  */
                                        /* (changes after each Tx DevIoctl   */
                                        /* to DD or after Tx completed)      */
  int                   StatusCount;    /* How many status events have been  */
                                        /* triggered.                        */
```

```
  UCHAR                 V24In;          /* Last 'getv24i/f' value got        */
  UCHAR                 V24Out;         /* Last 'setv24 outputs' value set   */


/*       The values for the indexes into the link statistics array of the */
/*       various types of statistic. */

  int                   StatusArray[SA_Max_Stat];

}                       IR,
                      * PIR;

#else
typedef struct teifrec {

        USHORT    RxFrameCount;
        USHORT    TxMaxFrSizeNow;
        USHORT    StatusCount;
        UCHAR     V24In;
        UCHAR     V24Out;
        USHORT    StatusArray[CEDSTMAX];

        }TEIFREC;

typedef TEIFREC far * TEIFRPTR;
#endif

/*****************************************************************************/
/* Structure for the set link characteristics parameter block.               */
/*****************************************************************************/

#ifdef WIN32
typedef struct  _SLPARMS
{
        int     SLFrameSize;            /* max frame size on link - must be  */
                                        /* in range 270 to ?2K-ish           */
        LONG    SLDataRate;             /* not used by us - external clocks  */
        UCHAR   SLOurAddress1;          /* ) e.g C1/FF or 00/00 or 01/03     */
        UCHAR   SLOurAddress2;          /* )                                 */
        UCHAR   SLLinkOptionsByte;      /* see documentation & LinkOption_*  */
        UCHAR   SLSpare1;
}
                SLPARMS;
#else

typedef struct teslcrec {

        USHORT  SLFrameSize;
        ULONG   SLDataRate;
        UCHAR   SLOurAddress1;
        UCHAR   SLOurAddress2;
        UCHAR   SLLinkOptionsByte;
        UCHAR   SLSpare1;

        }TESLCREC;

#endif

/*****************************************************************************/
/* DEVICEIOCTL macros                                                        */
/*****************************************************************************/
#ifdef WIN32
/* NT_SUCCESS ripped of from DDK's ntdef.h, which we do not want to include  */
/* for now temporarily (12/5/92)                                             */
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)

#define SETEVENTHANDLE(H) NtDeviceIoControlFile(                  \
                seldrvrh,                   \
```

```
                H,                          \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                &IoStatus,                  \
                IoctlCodeSetEvent,          \
                ( PVOID ) NULL,             \
                0L,                         \
                ( PVOID ) NULL,             \
                0L                          \
               )

#define SETINTERFACERECORD(R) NtDeviceIoControlFile(   \
                seldrvrh,                   \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                &IoStatus,                  \
                IoctlCodeSetInterfaceRecord,           \
                                            \
                &R,                         \
                sizeof(R),                  \
                ( PVOID ) NULL,             \
                0L                          \
               )

#define SETV24STATUS NtDeviceIoControlFile(            \
                seldrvrh,                   \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                &IoStatus,                  \
                IoctlCodeSetV24,            \
                NULL,             \
                0L,                         \
                &pInterfaceRecord->V24Out,  \
                1L                          \
               )


/*****************************************************************************/
/* The above change is temporary!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*****************************************************************************/

#define GETV24STATUS NtDeviceIoControlFile(   \
                  seldrvrh,                   \
                  ( PVOID ) NULL,             \
                  ( PVOID ) NULL,             \
                  ( PVOID ) NULL,             \
                  &IoStatus,                  \
                  IoctlCodeGetV24,            \
                  ( PVOID ) NULL,             \
                  0L,                         \
                  ( PVOID ) NULL,             \
                  0L                          \
                 )

#define SETLINKCHARACTERISTICS(A) NtDeviceIoControlFile(      \
                seldrvrh,                   \
                NULL,                       \
                ( PVOID ) NULL,             \
                ( PVOID ) NULL,             \
                &IoStatus,                  \
                IoctlCodeSetLinkChar,       \
                &A,                         \
                sizeof(A),                  \
                ( PVOID ) NULL,             \
                0L                          \
               )
```

```
#define TRANSMITFRAME(A,B) NtDeviceIoControlFile(            \
                 seldrvrh,                 \
                 NULL,                     \
                 ( PVOID ) NULL,           \
                 ( PVOID ) NULL,           \
                 &IoStatus,                \
                 IoctlCodeTxFrame,         \
                 ( PVOID ) NULL,           \
                 0L,                       \
                 A,                        \
                 B                         \
                )

#define RECEIVEFRAME(A,B) NtDeviceIoControlFile(             \
                 seldrvrh,             \
                 NULL,                 \
                 ( PVOID ) NULL,       \
                 ( PVOID ) NULL,       \
                 &IoStatus,            \
                 IoctlCodeRxFrame,     \
                 ( PVOID ) NULL,       \
                 0L,                   \
                 A,                    \
                 B                     \
                )

#define READINTERFACERECORD NtDeviceIoControlFile( \
                 seldrvrh,             \
                 NULL,                 \
                 ( PVOID ) NULL,       \
                 ( PVOID ) NULL,       \
                 &IoStatus,            \
                 IoctlCodeReadInterfaceRecord, \
                 ( PVOID ) NULL,       \
                 0L,                   \
                 &InterfaceRecord,     \
                 sizeof(InterfaceRecord) \
                )

#else
#define NT_SUCCESS(R) ((R) == 0)

#define SETEVENTHANDLE(H)  DosDevIOCtl(NULL,           \
                   &H,                             \
                   IoctlCodeSetEvent,              \
                   CELDDCAT,                       \
                   seldrvrh)

#define GETINTERFACERECORD(P) DosDevIOCtl(NULL,        \
                   &P,                             \
                   IoctlCodeGetInterfaceRecord,    \
                   CELDDCAT,                       \
                   seldrvrh)

#define SETV24STATUS DosDevIOCtl(NULL,                 \
                   NULL,                           \
                   IoctlCodeSetV24,                \
                   CELDDCAT,                       \
                   seldrvrh)

#define GETV24STATUS DosDevIOCtl(NULL,                 \
                     NULL,                         \
                     IoctlCodeGetV24,              \
                     CELDDCAT,                     \
                     seldrvrh)

#define SETLINKCHARACTERISTICS(A) DosDevIOCtl((long) NULL,  \
```

```
                    (char far *) &A,                        \
                    IoctlCodeSetLinkChar,                   \
                    CELDDCAT,                               \
                    seldrvrh)

#define TRANSMITFRAME(F,L) DosDevIOCtl(F,               \
                    &L,                                 \
                    IoctlCodeTxFrame,                   \
                    CELDDCAT,                           \
                    seldrvrh)

#define RECEIVEFRAME(F,L)  DosDevIOCtl(F,               \
                           &L,                          \
                           IoctlCodeRxFrame,            \
                           CELDDCAT,                    \
                           seldrvrh)
#endif

//############################################################################

/*****************************************************************************/
/* INFO_ : additional information error codes put in IoStatus.Information    */
/*****************************************************************************/

#define INFO_CANT_ALLOCATE_SPINLOCK      1
#define INFO_CANT_CONNECT_INTERRUPT      2
#define INFO_HARDWARE_INIT_FAILURE       3
#define INFO_SET_EVENT_NO_EVENT          4
#define INFO_HARDWARE_CMD_TIMEOUT        5
#define INFO_LINKCHAR_BUF_WRONG_SIZE     6
#define INFO_FRAME_BUF_TOO_BIG           7
#define INFO_FRAME_BUF_TOO_SMALL         8
#define INFO_NO_CLOCKS                   9
#define INFO_NO_DMA_FDX                 10
#define INFO_CANT_ALLOCATE_MDL          11
#define INFO_CANT_ALLOCATE_MEMORY       12
#define INFO_DMA_BUFFER_UNUSABLE        14
#define INFO_TX_BUFFER_FULL             15
#define INFO_TX_FRAME_TOO_BIG           16
#define INFO_TX_FRAME_TOO_SMALL         17
#define INFO_READ_IR_BUFFER_WRONG_SIZE  18
#define INFO_NEEDS_MCA_BUS              19
#define INFO_NEEDS_ISA_BUS              20
```

SNA Modem Status Interface

This section describes the interface to the modem status used by Synchronous Data Link Control (SDLC) and X.25 SNALinks that Host Integration Server 2009 supplies. This interface provides a set of simulated modem lights to show modem status. The modem status interface is intended primarily for the Microgate cards with internal modems, but can be used with any SDLC or X.25 link service to show the modem status.

This interface is intended primarily for independent hardware vendors (IHV) who want to provide drivers for their own dumb cards that are directly compatible with SDLC and X.25 SNALinks that Host Integration Server provides. It can also be used by IHVs who intend to write their own SNALinks but who want to ensure that their drivers conform to the standard Host Integration Server model.

In This Section

SNA Device Driver Interface to Modem Status

There are three interfaces from which modem status can be gathered:

- The SNADIS Dumb Card Interface **GetV24Status** IOCTL call that reports the status of the Ring Indicator (DRI), Carrier Detect (DCD), Clear To Send (CTS), and Data Set Ready (DSR) signal lines.

- The SNADIS Dumb Card Interface **SetV24Status** IOCTL call that sets the status of the Data Terminal Ready (DTR) and Request To Send (RTS) signal lines.

- The SNADIS Dumb Card Interface receive and transmit IOCTL calls, used to set the operating mode of the device driver.

Whenever one of the first two interfaces indicates a modem line is high, the corresponding light in the display is lit. However, the transmit and receive IOCTL calls cannot provide a definitive statement as to whether the card is receiving data, only that it is ready to receive data. To work around this limitation and to have the modem lights give a reasonable indication of ordinary data throughput, the following mechanism is recommended:

Simulate the receive and transmit lights with counters and track the number of frames received and transmitted for each link service. The display application uses this information to control the flashing of the receive and transmit lights.

The other topics in this section describe the changes that must be made to the link service for supporting the modem status interface and the interface provided to the display application. Microsoft Host Integration Server 2009 comes with an application that displays the modem status.

Supporting Modem Status in an SNA Link Service

The implementation of a link service that supports the modem status requires the following:

- A call to the SNA Modem interface to initialize the SNA Modem data structures.
- Modification of the SNA Modem structures as the status changes, as reported by the **DevIoctl** calls.

Independent hardware vendors (IHVs) who use the MicrosoftHost Integration Server 2009 SDLC link service and who fully implement the SNADIS interface do not need to make any changes to use the modem status feature. However, IHVs should check that the status returned by **DevIoctl** calls from their device driver conforms to the requirements described later.

IHVs who provide both the device driver and link service need to implement the code in their link service that initializes the SNA Modem API and updates the modem status information.

Modem API Summary

To simplify the task for independent hardware vendors (IHVs) who want to use this feature, four new entry points have been added to SNALINK.DLL. An IHV who uses these must be linking with IHVLINK.LIB, a stub library that contains the exports library for SNALINK.DLL. This API enables the IHV to simply maintain the contents of a MODEM_STATUS structure. The underlying SNALINK library code handles the communication of this information to the modem lights application.

The modem status functions are as follows.

Function	Description
SNAModemInitialize	Initializes the communication path to the SNA Modem application.
SNAModemAddLink	Adds an SNA link.
SNAModemDeleteLink	Deletes the resources associated with a link.
SNAModemTerminate	Terminates an SNA link.

DevIoctl Definitions to Support SNA Modem Status

The **SNA DevIoctl** interface is modified to update the MODEM_STATUS structure for a link each time a modem status change is detected or caused by a **GetV24** or **SetV24** IOCTL call. Code is manually added to the link service to track the number of frames received and transmitted.

The **DevIoctl** changes are highlighted as follows.

```
#define SETV24STATUS                                                \
         NtDeviceIoControlFile(seldrvrh,NULL,NULL,NULL,&IoStatus,  \
                               IoctlCodeSetV24,NULL,0L,             \
                               &pInterfaceRecord->V24Out,1L);       \
         if (SavedIROut != (InterfaceRecord.V24Out &                \
                            (MASK_DTR | MASK_RTS)    ))            \
         {                                                          \
           SavedIROut = (pInterfaceRecord->V24Out &                 \
                         (MASK_DTR | MASK_RTS)    );                \
           pSharedMem->V24Out = pInterfaceRecord->V24Out;           \
         }


#define GETV24STATUS(rc)                                            \
         rc |= NtDeviceIoControlFile(seldrvrh,NULL,NULL,NULL,       \
             &IoStatus,IoctlCodeGetV24,NULL,0L,NULL,0L);            \
         rc |= READINTERFACERECORD;                                 \
         if (SavedIRIn != (InterfaceRecord.V24In &                  \
                     (MASK_CTS | MASK_DSR | MASK_DCD| MASK_DRI))) \
         {                                                          \
             SavedIRIn = (InterfaceRecord.V24In &                   \
                     (MASK_CTS | MASK_DSR | MASK_DCD| MASK_DRI)); \
             pSharedMem->V24In = InterfaceRecord.V24In;             \
         }
```

pSharedMem is a pointer to the MODEM_STATUS structure for this link service.

V24In and *V24Out* are of type char and are used to notify the display application when status changes, not each time it is read or set.

SNA Performance Monitor Interface

This section describes the interface for performance monitoring (Perfmon) used by SNADIS links that Microsoft Host Integration Server 2009 supplies. This interface is provided to simplify the integration of SNADIS-compliant link services with the Microsoft Windows 2000 System Monitor applications. It provides a common look-and-feel to all link service performance counters exported by SNADIS links, independent of the vendor and link transport (channel, Twinax, SDLC, X.25, TR, E/Net, and so on).

The performance monitoring statistics maintained for an SNA link service are stored in a series of ADAPTERCOUNTER structures that are members of an ADAPTERPERFDATA structure. These structures are defined in the SEMFPERF.H header file.

Three API entry points are exported from IHVLINK.DLL (and the IHVLINK.LIB import library) that are used by the Perfmon API. These functions should be called in the order noted below at link service initialization time.

To support performance monitoring, an SNA Link driver first calls SNAInitLinkPerfmonto initialize data structures used by the Perfmon application. This call should be followed with a call to the function SNAGetLinkPerfArea, which returns a shared mutex handle and a pointer to the shared data area for the **ADAPTERPERFDATA** structure used by the Perfmon application to store the link statistics. This handle and shared memory data area parameter are the returned values from **SNAInitLinkPerfmon**. Finally, the SNAGetPerfValues function is called to fill in the *ServiceNameIndex* and *FirstCounterIndex* fields so that the Perfmon application knows where to get the descriptions of the performance counters from the registry.

After these three calls have been made, the SNA link driver simply maintains the count members in the **ADAPTERCOUNTER** structures that make up the **ADAPTERPERFDATA** structure, incrementing the count member whenever data is received, connections fail, and other events occur. The Perfmon application accesses these counters to display Host Integration Server performance monitoring data statistics.

Network Integration Security Guides

The network integration features of Microsoft Host Integration Server 2009 fall into several areas. Each area has a Security guide.

This section contains:

- APPC Programmer's Security Guide
- CPI-C Programmer's Security Guide
- LUA Programmer's Security Guide
- SNA Print Server Data Filter Programmer's Security Guide

APPC Programmer's Security Guide

The following topics discuss security as it applies to the APPC section of the Microsoft Host Integration Server 2009 SDK.

About APPC security for developers

This section discusses security practices and issues that apply to C-language applications that you write to use Advanced Program-to-Program Communications (APPC) to exchange data in a Systems Network Architecture (SNA) environment.

Conversation Security

- Owners of APPC transaction programs may want to allow only a limited set of users to start the program. APPC provides a mechanism, called APPC conversation security, where the client transaction program supplies credentials to the server to gain access to the program. You can find more information on conversation security in the section: APPC Security.

Several APPC verbs create a connection with a remote LU. When creating a connection with a remote LU, credentials may be required to establish access. The **security** parameter controls conversation security, and the **pwd** and **user_id** parameters specify the information used to validate the user on the host.

Conversational security is documented in the topic Conversation Security, and in the reference pages for the following APPC verbs:

- ALLOCATE
- MC_ALLOCATE
- SEND_CONVERSATION
- MC_SEND_CONVERSATION

Session Security configuration using SNACFG

You can use the utility SNACFG to set session security for a remote LU. Possible values are none, use a plaintext key, and use a scrambled key.

Automatic Login

The configuration of Host Integration Server to support automatic login is discussed in the topic Support for CPI-C Automatic Logon.

Setting the service key with CNOS

You can specify a master or service key using the **key** parameter when changing the number of sessions with the CNOS verb. This is a plaintext parameter, and is valid if the keylock feature has been secured.

CPI-C Programmer's Security Guide

The following topics discuss security as it applies to the CPI-C section of the Host Integration Server 2009 SDK.

About CPI-C Security for Developers

The Common Programming Interface for Communications (CPI-C) is a platform-independent API designed to simplify the use APPC verbs, and has many of the same issues and features as APPC.

This section discusses security practices and issues that apply to C-language applications that you write to use CPI-C to exchange data between a Host Integration Server system and a computer or computers in a Systems Network Architecture (SNA) environment.

Session Security Configuration Using SNACFG

You can use the utility SNACFG to set session security for a remote LU. Possible values are none, use a plaintext key, and use a scrambled key.

Conversation Security in CPI-C

Conversation security in CPI-C controls who can connect to a remote LU. You can find more information in the topic Conversation Security.

A programmer controls conversation security using the Set_Conversation_Security_Type call and sets user credentials using the Set_Conversation_Security_User_ID and Set_Conversation_Security_Password call from the CPI-C SDK.

LUA Programmer's Security Guide

The following topics discuss security as it applies to the LUA section of the Host Integration Server 2009 SDK.

About LUA security for developers

LUA presents a simple API that enables programmatic control over the SNA messages. LUA has a simple structure, but exposes a complex data stream, that associated with SNA messages.

Since LUA is a low-level interface, the programmer must be especially vigilant on security issues.

Threats and mitigations for LUA

Programmers who use this feature should be aware of the following security practices and issues.

The data stream between Host Integration Server and the Mainframe is not encrypted.

LUA provides no encryption for the data stream between Host Integration Server and the Mainframe.

SNA Print Server Data Filter Programmer's Security Guide

The following topics discuss security as it applies to the APPC section of the Host Integration Server 2009 SDK.

Threats and mitigations for SNA Print Server Data Filter

Programmers who use the SNA Print Server Data Filter should be aware of the following security issue.

Store DLLs in a secure location

The DLLs which support the SNA Print Server Data Filter API can be configured by an administrator or developer. Because of this, the DLLs should be stored in a secure location, such as the default installation directory. If this location is customized by an administrator or developer the new location should also be secured.

Session Integrator Programmer's Guide

This section describes how to create applications that interact with a remote server using LU0 and LU2 protocols.

In This Section

Session Integrator

Using Session Integrator

Reference

Microsoft.HostIntegration.SNA.Session

Session Integrator

An enterprise organization might run the bulk of their daily operations on an IBM host system, such as the zSeries IBM mainframe. Many host-based applications use session-based termination emulation and other client-server programs that use either Logical Unit type 0 (LU0) or Logical Unity type 2 (LU2). LU0 programs, common in retail banking, are based on the Logical Unit for Applications (LUA) programming model. In contrast, the LU2 programs, common in many vertical markets, are based on terminals and terminal emulation using a 3270 data stream.

Session Integrator provides COM and .NET Framework access to LU0 and LU2 sessions, using Host Integration Server as the gateway and platform for IBM interoperability.

In This Section

Session Integrator Programming Interfaces

What You Should Know Before Using Session Integrator

Supported Platforms for Session Integrator

COM Security Requirements for Session Integrator

Reference

Session Integrator COM Reference

Microsoft.HostIntegration.SNA.Session

Related Sections

Using Session Integrator

See Also

Other Resources

3270 Emulation Programmer's Reference

3270 Emulation Programmer's Guide

Session Integrator Programming Interfaces

Session Integrator provides a COM and a .NET Framework interface to grant programmatic access to an SNA network using LU0 and LU2 protocols. The following tables describe the relationships between the COM and the .NET Framework interfaces.

LU0 Interfaces

COM	.NET	Description
IcomLU0	SessionLU0	The primary interface for connecting, sending, and receiving information over an LU0 session.
	SessionConnectionLU0	Represents the LU0 connection. **SessionLU0** uses **SessionConnectionLU0** to contain the relevant connection and initialization information.
	SessionLU0Data	Encapsulates the information you want to send and receive over LU0. **SessionLU0** uses **SessionLU0Data** in the **Send** and **Receive** methods to send and receive data.
	SessionLU0InitType	Contains initialization information. **SessionLU0** uses this class during the initialization process to pass initialization information to the host.
	SessionLU0ReceiveIndication	Represents the current state of a session associated with a receive call.
	SessionLU0SendIndication	Contains values used by the **Send** method of **SessionLU0**.

LU2 Interfaces

COM	.NET	Description
	SessionDisplay	Provides the connection interface for the **SessionDisplay** class.
	ScreenPosition	Provides access to a position on the LU2 screen.
	ScreenCursor	Provides access to the cursor on the screen.
	ScreenPartialField	Provides access to a part of a screen field.
	ScreenField	Provides access to a particular area of the LU2 screen including the data and attributes.
	ScreenFieldCollection	Contains a collection of **ScreenField** classes.
	ScreenPartialFieldCollection	Contains a collection of **ScreenPartialFields** classes.
	ScreenFieldAttributeData	Provides all of the attributes about the **ScreenField** data.
Icom3270	**SessionDisplay**	Primary interface for accessing the network over a 3270 connection.
	ScreenData	Provides access to a particular area of the raw 3270 data representations.
	SessionDisplayScript	Enables users to take a script created in the Host Integration Server 3270 client and play it programmatically.

	SessionDisplayVariableCollection	Used with the **SessionDisplayScript** class to provide variables that can be replaced in the script.

See Also

Other Resources

Session Integrator

Using Session Integrator

What You Should Know Before Using Session Integrator

Session Integrator is designed to communicate between a Windows-based enterprise and an IBM network. Therefore, you should be familiar with the following concepts and technologies before attempting to use Session Integrator:

- Programming with the .NET Framework or COM
- Windows Networking
- SNA Networking

See Also

Other Resources

Session Integrator

Using Session Integrator

Supported Platforms for Session Integrator

Session Integrator is designed to be used with Host Integration Server 2009, and is therefore supported on all platforms that support Host Integration Server 2009.

See Also

Other Resources

Session Integrator

Using Session Integrator

COM Security Requirements for Session Integrator

When using Session Integrator in an environment where the Client application and the Service component are running under two different user accounts, special configuration of COM security should be performed.

To configure COM security in Component Services

1. Click Start, then click Run, then type **DCOMCNFG**, and then click **OK**.
2. In Component Services, expand **Component Services**, expand **Computers**, right-click My **Computer**, and then click **Properties**.
3. In **My Computer Properties**, in the **COM Security** tab, under **Access Permissions**, click **Edit Default**.
4. In **Access Permission** dialog box, click **Add** to add the Server user context. If the Server is not on the same computer as the client, you must allow Remote Access to the user.
5. Click **OK** twice.

Using Session Integrator

The topics in this section describe how to create Session Integrator applications.

In This Section

Using Session Integrator for LU0

Using Session Integrator for LU2

Using Session Integrator for TN3270

Using Session Integrator for LU0

This section describes how to create an application by using Session Integrator over an LU0 connection.

In This Section

How to Initialize a Session Integrator Session for LU0

How to Send a Message Using Transaction Integrator for LU0

How to Receive a Message Using Transaction Integrator for LU0

How to Terminate a Connection with Session Integrator for LU0

Using Pre-Recorded Scripts with Session Integrator

Session Integrator for LU0 Code Example

Reference

Microsoft.HostIntegration.SNA.Session

IcomLU0 Interface

Related Sections

Using Session Integrator

See Also

Other Resources

LUA Programmer's Guide

How to Initialize a Session Integrator Session for LU0

The first action you must perform when you connect to an LU0 session for Session Integrator is to create and initialize the SessionLU0 object. As the name implies, **SessionLU0** represents the LU0 session to your application, and is the primary interface that you will use to access the SNA network.

After you initialize the connection, you can begin to send and receive information over the LU0 session.

To initialize a Session Integrator session for LU0

1. Determine what type of session you will connect to.
2. If necessary, create a new session connection by using SessionConnectionLU0.

   You can create the **SessionConnectionLU0** directly if you have all the relevant information. However, you do not need to perform this step. More likely, you will just pass in the LU connection string in step 3.
3. Create a new session by using **SessionLU0**.
4. Pass the connection information to Connect.

   Connect contains several overloads: you can choose to connect with an already-created SessionConnection object, a **SessionConnection** object and additional initialization information, or with a connection string and initialization information.

   If you choose to call **Connect** with a connection string, Session Integrator creates a new **SessionConnectionLU0** for you. You can directly access the **SessionConnectionLU0** object through Connection.
5. If necessary, confirm that you connected by using IsConnected.

Example

The following code example demonstrates how to create a session, using a connection string received from the user.

```
private void CreateSession_Click(object sender, EventArgs e)
 {
   try
     {
      LUName.Text = LUName.Text.Trim();
      if (LUName.Text.Length == 0)
       {
       MessageBox.Show("You must fill out the LU or Pool Name");
       return;
       }
    _session = new SessionLU0();
    _session.Connect("LogicalUnitName=" + LUName.Text, SessionLU0InitType.SSCP);
                // Receive the logon screen.
    SessionLU0Data receivedData = _session.Receive(20000, true);
                // Trace out the received data.
    TraceData(false, receivedData.Data, receivedData.Indication);
                // Disable every button and text box.
    DisableEverything();
                // Insert User/Password.
    EnableInsertUserId();
   }
catch (Exception ex)
 {
  MessageBox.Show(ex.Message);
 }
}
```

Note that the primary purpose of this code example is to create a new session and connect to the LU using a connection string. However, the example also receives data back over the LU0 session. The example also sends password information using the **EnableInsertUserId** function.

See Also

Tasks

How to Send a Message Using Transaction Integrator for LU0
Session Integrator for LU0 Code Example

Reference

Microsoft.HostIntegration.SNA.Session

Concepts

IcomLU0 Interface

Other Resources

Using Session Integrator for LU0

How to Send a Message Using Transaction Integrator for LU0

After you initialize and connect to the logical unit (LU), you can send information over the LU0 connection. The primary tool that Session Integrator exposes for sending LU0 information is the **SessionLU0Data** object and the **SessionLU0.Send** method.

In addition to sending information, you probably will want to receive information too.

To send a message using Transaction Integrator for LU0

1. Collect your data into the format your LU uses.
2. Place your data into a **SessionLU0Data** object.
3. Send the data with **SessionLU0.Send**.

Example

The following code example demonstrates how to send data over an LU0 session using Session Integrator.

```
private void InsertUserId_Click(object sender, EventArgs e)
 {
  try
   {
    // Disable every button and text box.
     DisableEverything();
     // Enter UserName (SNA200 is what is in the script).
     // AID = 7D - Enter.
     byte AID = 0x7D;
     // Cursor address.
     byte ca1 = 0x5B;
     byte ca2 = 0x6B;
     // SBA
     byte SBA = 0x11;
     byte fa1 = 0x5B;
     byte fa2 = 0xE5;
     byte[] sna200 = HostStringConverter.ConvertUnicodeToEbcdic("SNA200");
     byte sixD = 0x6D;
     byte [] message = new byte [8 + sna200.Length ];
     message[0] = AID;
     message[1] = ca1;
     message[2] = ca2;
     message[3] = SBA;
     message[4] = fa1;
     message[5] = fa2;
     Array.Copy(sna200, 0, message, 6, sna200.Length);
     message[6 + sna200.Length] = sixD;
     message[7 + sna200.Length] = sixD;
     // Send the data.
     SessionLU0Data data = new SessionLU0Data();
     data.Data = message;
     // Trace out the data to send.
     TraceData(true, message, 0);
     _session.Send(data);
     // Allow entering director.
     EnableEnterDirector();
     }
    catch (Exception ex)
    {
     MessageBox.Show(ex.Message);
    }
   }
```

Most of this code example is about formatting the data so that the LU can correctly interpret the information; the call to **SessionLU0.Send** is relatively simple. For more information about the code sample, see Session Integrator for LU0 Code Example.

See Also

Tasks

How to Receive a Message Using Transaction Integrator for LU0

Session Integrator for LU0 Code Example

Reference

Microsoft.HostIntegration.SNA.Session

Concepts

IcomLU0 Interface

Other Resources

Using Session Integrator for LU0

How to Receive a Message Using Transaction Integrator for LU0

After you create and initialize your connection, you can receive information from the specified logical unit (LU). The primary way of receiving information with Session Integrator is with the **SessionLU0.Receive** method.

After sending and receiving messages, you must disconnect from your Session Integrator session.

To receive information using Session Integrator for LU0

1. Use SessionLU0.Receive and **SessionLU0data** to wait for data from the LU.

   Receive allows you to pass the maximum length of time to wait for information, as well as whether you want to send an automatic acknowledgement. **Receive** returns a **SessionLU0Data** object.

Example

The following code example demonstrates how to receive information with Session Integrator for LU0.

```
private void CreateSession_Click(object sender, EventArgs e)
 {
   try
     {
      LUName.Text = LUName.Text.Trim();
      if (LUName.Text.Length == 0)
       {
       MessageBox.Show("You must fill out the LU or Pool Name");
       return;
       }
    _session = new SessionLU0();
    _session.Connect("LogicalUnitName=" + LUName.Text, SessionLU0InitType.SSCP);
                // Receive the logon screen.
    SessionLU0Data receivedData = _session.Receive(20000, true);
                // Trace out the received data.
    TraceData(false, receivedData.Data, receivedData.Indication);
                // Disable every button and text box.
    DisableEverything();
                // Insert User/Password.
    EnableInsertUserId();
   }
catch (Exception ex)
 {
  MessageBox.Show(ex.Message);
 }
}
```

For more information about the code sample, see Session Integrator for LU0 Code Example.

See Also

Tasks

How to Terminate a Connection with Session Integrator for LU0

Session Integrator for LU0 Code Example

Reference

Microsoft.HostIntegration.SNA.Session

Concepts

IcomLU0 Interface

Other Resources

Using Session Integrator for LU0

How to Terminate a Connection with Session Integrator for LU0

After you finish sending and receiving information over the LU0 session, you must terminate your connection. Ending a session with Session Integrator consists primarily of calling **SessionLU0.Disconnect**.

To terminate a connection with Session Integrator for LU0

1. End the connection by using **SessionLU0.Disconnect**.

   After you call **Disconnect**, you must call **Connect** in order to re-establish a connection. This is true even if the call fails to complete successfully.

2. If necessary, ensure that you have released all relevant data objects.

Example

The following code example shows how to terminate a Session Integrator session for LU0.

```
private void Disconnect_Click(object sender, EventArgs e)
 {
 // Disable every button and text box.
  DisableEverything();
  _session.Disconnect();
// Go back to the original state of buttons.
  SetOpeningState();
  }
```

See Also

Other Resources

Using Session Integrator for LU0

Using Pre-Recorded Scripts with Session Integrator

The **SessionDisplayScript** class allows users to use a script created in the Host Integration Server 3270 client and play it programmatically.

The script can implement variables using a double percent sign on each end of the name, for example, %%MYVARIABLE%%. These variables are resolved using the SessionDisplayVariableCollection class provided in this class. In addition, the script file can contain environment variables using the standard notation which this class will translate.

SessionDisplayScript Class

The input script must be a normal text file with one command per line. The Script file supports the following commands:

Command	Description
SETTIMEOUT {timeout},{label}	Sets the default timeout for all commands and the label where processing should continue. If no default is set, 30 seconds is assumed.
WAITSESSION {wait}	Waits for the session to be in the input wait state before returning. The accepted values are: SSCP; LULU; UNOWNED
WAIT {seconds}	Waits the input number of seconds and then moves to the next command. The WAIT command can be replaced by the WAITSTRING command to wait for a specific string on the screen.
SETCURSOR {ROW},{COLUMN}	Moves the cursor to the desired position on the screen. If the position is not found on the screen, the script is aborted and a ScriptError Exception is returned with an InnerException of the actual Exception when running the script.
SEND {string}, {%environmentvariable%}, {%%sessiondisplayvariable%%}	Causes the string to be sent to the screen using the SendKeys method. Variables can be input that are matched against the SessionDisplayVariablesCollection passed into the class. If a variable is not located in the script, the script is aborted and a ScriptError Exception is returned with a InnerException of Variable {name} not located in collection.
GOTO {label}	Allows for scripts to jump to labels below the current line. If the label is not found, the script will abort with a ScriptError exception and an InnerException of “Label {name} not found”. {label} = A way to define a freeform label in the script that can be used in branching scenarios.

Session Integrator for LU0 Code Example

The following code example shows how to use the primary techniques for creating an LU0 connection, logging on to the LU0 session, sending and receiving information, and terminating the connection.

For the complete code sample, see the \\Microsoft Host Integration Server\SDK\Samples\AppInt\COMLU0 directory.

Example

The following example is from the CSClient.Form1 file in the COMLU0 sample.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.HostIntegration.SNA.Session;

namespace CSClient
{
    public partial class Form1 : Form
    {
        private SessionLU0 _session = null;

        // As the LU0 managed wrapper does no tracing
        // we will trace the data contents to the provided text box.
        private TextBox m_TextBox = null;
        private Font m_FixedFont;

        public Form1()
        {
            InitializeComponent();
        }

        private void SetOpeningState()
        {
            DisableEverything();

            // Only the LU Name and Create Session are enabled.
            this.CreateSession.Enabled = true;
            this.LUName.Enabled = true;

            m_TextBox = this.ScreenText;

            // If we should trace, we need a fixed width font.
            FontFamily fontFamily = new FontFamily("Courier New");
            m_FixedFont = new Font(fontFamily, 10, FontStyle.Regular, GraphicsUnit.Pixel);

            // Set up some things.
            m_TextBox.WordWrap = false;
            m_TextBox.Multiline = true;

            // Find a fixed font.
            m_TextBox.Font = m_FixedFont;
        }

        private void DisableEverything()
        {
            // All buttons are disabled.
            this.CreateSession.Enabled = false;
            this.InsertUserId.Enabled = false;
            this.EnterDirector.Enabled = false;
            this.Disconnect.Enabled = false;

            // All text boxes are disabled.
```

```
            this.LUName.Enabled = false;
        }

        private void EnableDisconnect()
        {
            // Just allow the Disconnect.
            this.Disconnect.Enabled = true;
        }

        private void EnableInsertUserId()
        {
            // Enable the cics name / connect.
            this.InsertUserId.Enabled = true;

            // Enable the disconnect.
            EnableDisconnect();
        }

        private void EnableEnterDirector()
        {
            // Enable clear screen.
            this.EnterDirector.Enabled = true;

            // Enable the disconnect.
            EnableDisconnect();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Enable only the LU Name and Create.
            SetOpeningState();
        }

        private void CreateSession_Click(object sender, EventArgs e)
        {
            try
            {
                LUName.Text = LUName.Text.Trim();
                if (LUName.Text.Length == 0)
                {
                    MessageBox.Show("You must fill out the LU or Pool Name");
                    return;
                }

                _session = new SessionLU0();
                _session.Connect("LogicalUnitName=" + LUName.Text, SessionLU0InitType.SSCP)
;

                // Receive the logon screen.
                SessionLU0Data receivedData = _session.Receive(20000, true);

                // Trace out the received data.
                TraceData(false, receivedData.Data, receivedData.Indication);

                // Disable every button and text box.
                DisableEverything();

                // Insert User/Password.
                EnableInsertUserId();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void InsertUserId_Click(object sender, EventArgs e)
        {
```

```
        try
        {
            // Disable every button and text box.
            DisableEverything();

            // Enter UserName (SNA200 is what is in the script).
            // AID = 7D - Enter
            byte AID = 0x7D;
            // Cursor address.
            byte ca1 = 0x5B;
            byte ca2 = 0x6B;
            // SBA
            byte SBA = 0x11;
            byte fa1 = 0x5B;
            byte fa2 = 0xE5;

            byte[] sna200 = HostStringConverter.ConvertUnicodeToEbcdic("SNA200");

            byte sixD = 0x6D;

            byte [] message = new byte [8 + sna200.Length ];
            message[0] = AID;
            message[1] = ca1;
            message[2] = ca2;
            message[3] = SBA;
            message[4] = fa1;
            message[5] = fa2;

            Array.Copy(sna200, 0, message, 6, sna200.Length);

            message[6 + sna200.Length] = sixD;
            message[7 + sna200.Length] = sixD;

            // Send the data.
            SessionLU0Data data = new SessionLU0Data();
            data.Data = message;

            // Trace out the data to send.
            TraceData(true, message, 0);

            _session.Send(data);

            // Allow entering director.
            EnableEnterDirector();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void EnterDirector_Click(object sender, EventArgs e)
    {
        try
        {
            // Disable every button and text box.
            DisableEverything();

            // Receive the Director screen.
            SessionLU0Data receivedData = _session.Receive(20000, true);

            // Trace out the received data.
            TraceData(false, receivedData.Data, receivedData.Indication);

            EnableDisconnect();
        }
        catch (Exception ex)
        {
```

```
                MessageBox.Show(ex.Message);
            }
        }

        private void Disconnect_Click(object sender, EventArgs e)
        {
            // Disable every button and text box.
            DisableEverything();

            _session.Disconnect();

            // Go back to the original state of buttons.
            SetOpeningState();
        }

        // Print out the Data to a provided text box.
        private void TraceData(bool sent, byte[] data, int indication)
        {
            if (m_TextBox == null)
                return;

            // Was the last thing sent or received?
            if (sent)
                m_TextBox.Text += "====>> Sent to Host" + Environment.NewLine;
            else
                m_TextBox.Text += "<<==== Received from Host" + Environment.NewLine;

            // How much is there to trace.
            int traceLength = data.Length;

            m_TextBox.Text += "Size = " + traceLength.ToString();

            if (!sent)
                m_TextBox.Text += String.Format(", Indication = {0:X}", indication);

            m_TextBox.Text += Environment.NewLine;

            int numberTraced = 0;
            while (numberTraced < traceLength)
            {
                string hexLine = "";
                for (int i = 0; i < 16; i++)
                {
                    if (numberTraced + i >= traceLength)
                        hexLine += "   ";
                    else
                        hexLine += String.Format("{0:x2} ", data[numberTraced + i]);
                }

                m_TextBox.Text += hexLine + Environment.NewLine;

                numberTraced += 16;
            }
        }
    }
}
```

See Also

Reference

Microsoft.HostIntegration.SNA.Session

Other Resources

Using Session Integrator for LU0

Using Session Integrator for LU2

This section describes how to program an application for Session Integrator using an LU2 connection.

In This Section

How to Initialize a Session Integrator Session for LU2

How to Send a Message Using Session Integrator for LU2

How to Receive a Message Using Session Integrator for LU2

How to Terminate a Connection with Session Integrator for LU2

Session Integrator for LU2 Code Example

Reference

Microsoft.HostIntegration.SNA.Session

See Also

Other Resources

Using Session Integrator

How to Initialize a Session Integrator Session for LU2

The first action that you must perform when you are connecting to an LU2 session for Transaction Integrator is to create and initialize the SessionDisplay object. As the name implies, **SessionDisplay** represents the 3270 display to your application, and is the primary interface that you will use to access the SNA network.

After you initialize your connection, you can begin to send and receive information over your LU2 session.

Procedure Title

1. If necessary, create a new session connection with SessionConnectionDisplay.

   You can create the **SessionConnectionDisplay** directly if you have all the relevant information. However, you do not need to perform this step. More likely, you will simply pass in the LU connection string in step 2.

2. Create a new session with **SessionDisplay**.

3. Pass the connection information to Connect.

   Connect contains several overloads: you can choose to connect with an already-created **SessionDisplay** object, a **SessionDisplay** object and additional initialization information, or with a connection string and initialization information.

   If you choose to call **Connect** with a connection string, Transaction Integrator will create a new **SessionConnectionDisplay** for you. You can directly access the **SessionConnectionDisplay** object through Connection.

4. If necessary, confirm that you connected using IsConnected.

Example

The following code is from the COM3270 application in the SDK sample directory.

```
private void CreateSession_Click(object sender, EventArgs e)
        {
            try
            {
                LUName.Text = LUName.Text.Trim();
                if (LUName.Text.Length == 0)
                {
                    MessageBox.Show("You must fill out the LU or Pool Name");
                    return;
                }
                m_Handler = new SessionDisplay();

                m_Handler.Connect("TRANSPORT=SNA;LOGICALUNITNAME=" + LUName.Text);
                m_Handler.Connection.HostCodePage = 37;

                FontFamily fontFamily = new FontFamily("Courier New");
                m_FixedFont = new Font(fontFamily, 10, FontStyle.Regular, GraphicsUnit.Pixe
l);
                ScreenText.Font = m_FixedFont;
                TraceScreen();

                // Disable every button and text box.
                DisableEverything();

                m_Handler.WaitForContent("TERM NAME", 20000);
                TraceScreen();

                // Enable Connect to CICS and Disconnect Session.
                EnableCICSElements();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

See Also

Tasks

Session Integrator for LU2 Code Example

Reference

SessionDisplay

Other Resources

Using Session Integrator for LU2

How to Send a Message Using Session Integrator for LU2

After you create a connection, you can send information over the LU2 connection to the remote display.

To send a message using Transaction Integrator for LU2

1. If necessary, move the cursor to the position that you want to write to on the screen by calling one of the **SessionDisplay.Move** methods.

   SessionDisplay contains a variety of MoveCursor, MoveNextField, MovePreviousField, and MoveToField overloads. These overloads enable you to move the cursor to different parts of the screen, depending on what information you provide.

   The **SessionDisplay.Move** methods are mirrored by a similar set of **SessionDisplay.Get** methods, which enable you to retrieve the location of the cursor, as well as the information contained in different fields on the screen.

2. Send information to the current cursor position using a call to **SessionHandler.sendKey**.

   sendKey sends a specified string to the location on the screen marked by the cursor. If no cursor location is available, **sendKey** sends the information to the default location.

Example

The following code is from the 3270 application in the SDK sample directory. In this sample, the developer assumes that the cursor is in the default location on the screen, and therefore does not confirm the cursor location.

```
private void ConnectCICS_Click(object sender, EventArgs e)
        {
            try
            {
                CICSName.Text = CICSName.Text.Trim();
                if (CICSName.Text.Length == 0)
                {
                    MessageBox.Show("You must fill out the CICS Name");
                    return;
                }
                // Disable every button and text box.
                DisableEverything();
                m_Handler.SendKey(CICSName.Text + "@E");
                TraceScreen();
                m_Handler.WaitForSession (SessionDisplayWaitType.PLUSLU, 5000);
                TraceScreen();
                m_Handler.WaitForContent(@"DEMONSTRATION", 20000);
                TraceScreen();
                // Enable clear screen.
                EnableClearScreen();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
```

See Also

Reference

SessionDisplay

Other Resources

Using Session Integrator for LU2

Using Session Integrator for LU0

How to Receive a Message Using Session Integrator for LU2

After you create an LU2 session, you can retrieve information and messages from the 3270 console through the ScreenData and SessionDisplay objects.

To receive information over an LU2 connection

1. If necessary, retrieve the entire screen as a screen dump using **ScreenData**.

   For most circumstances, retrieving all the information on the screen is not necessary. Instead, you can use the **SessionDisplay** object for most applications.

2. Get the location of the cursor with a call to ScreenCursor.

3. Optionally, you can get the location and information contained within different fields on the screen with a call to one of the GetField or GetFields methods, or the CurrentField property.

   GetField and **GetFields** both contain multiple overloads, allowing you to retrieve field information from the screen, depending on what information you provide. In contrast, **CurrentField** represents only the field the cursor is currently in.

4. Finally, you can receive field update information with a call to the various **SessionDisplay.Wait** methods.

Example

The following code is from the 3270 application in the Host Integration Server SDK. The sample uses **SessionDisplay.CurrentField.Data** to access the screen data.

```
private void PerformTX_Click(object sender, EventArgs e)
        {
            try
            {
                // Disable every button and text box.
                DisableEverything();

                m_Handler.SendKey("@E");
                TraceScreen();

                // Wait for screen to calm down.
                m_Handler.WaitForSession(SessionDisplayWaitType.NotBusy, 5000);
                TraceScreen();

                // See if the Balance Field is filled out.
                m_Handler.Cursor.Row = m_row;
                m_Handler.Cursor.Column = m_column;
                TraceScreen();
                // Tab to the Account Number field.
                m_Handler.SendKey("@T");
                TraceScreen();
                // Move to the Next Field (Empty Stuff after 123456).
                m_Handler.MoveNextField();
                TraceScreen();
                // Move to the Next Field (Title, Account Balance).
                m_Handler.MoveNextField();
                TraceScreen();
                // Move to the Next Field (Account Balance).
                m_Handler.MoveNextField();
                TraceScreen();

                // Extract Data from this field.
                string accountBalance = m_Handler.CurrentField.Data;

                // Trim the string.
                accountBalance = accountBalance.Trim();

                // Only things to do now are clear screen or disconnect.
                EnableClearScreen();
```

```
                // If we failed (not Abended) then this field will be blank.
                if (accountBalance.Length == 0)
                    throw new Exception("Failed to get Account Balance");
                else
                    MessageBox.Show(accountBalance, "Account Balance");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

See Also

Reference

ScreenData

SessionDisplay

Other Resources

Using Session Integrator for LU2

Using Session Integrator for LU0

How to Terminate a Connection with Session Integrator for LU2

After you have finished sending and receiving information on your LU2 connection, you must shut your connection down with a call to Disconnect.

To shut an LU2 connection down

1. When you are finished with your connection, call **SessionDisplay.Disconnect** to disconnect.

Example

The following code example is from the 3270 application in the Host Integration Server SDK.

```
private void Disconnect_Click(object sender, EventArgs e)
        {
            // Disable every button and text box.
            DisableEverything();

            m_Handler.Disconnect();

            // Go back to the original state of buttons.
            SetOpeningState();
        }
```

See Also

Reference

SessionDisplay

Other Resources

Using Session Integrator for LU2

Session Integrator for LU2 Code Example

The following code is from the 3270 application in the samples directory of the Host Integration Server SDK.

Example

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.HostIntegration.SNA.Session;

namespace CSClient
{
    public partial class Form1 : Form
    {
        private System.Drawing.Font m_FixedFont;
        private SessionDisplay m_Handler = null;
        private short m_row;
        private short m_column;

        public Form1()
        {
            InitializeComponent();
        }

        private void SetOpeningState()
        {
            DisableEverything();

            // Only the LU Name and Create Session are enabled.
            this.CreateSession.Enabled = true;
            this.LUName.Enabled = true;
        }

        private void DisableEverything()
        {
            // All Buttons are disabled.
            this.CreateSession.Enabled = false;
            this.ConnectCICS.Enabled = false;
            this.SendCCLI.Enabled = false;
            this.ClearScreen.Enabled = false;
            this.FillIPAddress.Enabled = false;
            this.FillPort.Enabled = false;
            this.PerformTX.Enabled = false;
            this.Disconnect.Enabled = false;

            // All Text Boxes are disabled.
            this.LUName.Enabled = false;
            this.CICSName.Enabled = false;
            this.IPAddress.Enabled = false;
            this.PortNumber.Enabled = false;
        }

        private void EnableDisconnect()
        {
            this.Disconnect.Enabled = true;
        }

        private void EnableCICSElements()
        {
            // Enable the cics name / connect.
            this.ConnectCICS.Enabled = true;
            this.CICSName.Enabled = true;
```

```
            // Enable the disconnect.
            EnableDisconnect();
        }

        private void EnableClearScreen()
        {
            // Enable clear screen.
            this.ClearScreen.Enabled = true;

            // Enable the disconnect.
            EnableDisconnect();
        }

        private void EnableCCLI()
        {
            // Enable Send CCLI.
            this.SendCCLI.Enabled = true;

            // Enable clear screen (and disconnect).
            EnableClearScreen();
        }

        private void EnableIPInfo()
        {
            // Enable IP Address, Port Number and Fill Buttons.
            this.IPAddress.Enabled = true;
            this.PortNumber.Enabled = true;
            this.FillIPAddress.Enabled = true;
            this.FillPort.Enabled = true;

            this.PerformTX.Enabled = true;

            // Enable clear screen (and disconnect).
            EnableClearScreen();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Enable only the LU Name and Create.
            SetOpeningState();
        }

        private void CreateSession_Click(object sender, EventArgs e)
        {
            try
            {
                LUName.Text = LUName.Text.Trim();
                if (LUName.Text.Length == 0)
                {
                    MessageBox.Show("You must fill out the LU or Pool Name");
                    return;
                }
                m_Handler = new SessionDisplay();

                m_Handler.Connect("TRANSPORT=SNA;LOGICALUNITNAME=" + LUName.Text);
                m_Handler.Connection.HostCodePage = 37;

                FontFamily fontFamily = new FontFamily("Courier New");
                m_FixedFont = new Font(fontFamily, 10, FontStyle.Regular, GraphicsUnit.Pixe
l);
                ScreenText.Font = m_FixedFont;
                TraceScreen();

                // Disable every button and text box.
                DisableEverything();

                m_Handler.WaitForContent("TERM NAME", 20000);
```

```
            TraceScreen();

            // Enable Connect to CICS and Disconnect Session.
            EnableCICSElements();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void ConnectCICS_Click(object sender, EventArgs e)
    {
        try
        {
            CICSName.Text = CICSName.Text.Trim();
            if (CICSName.Text.Length == 0)
            {
                MessageBox.Show("You must fill out the CICS Name");
                return;
            }

            // Disable every button and text box.
            DisableEverything();

            m_Handler.SendKey(CICSName.Text + "@E");
            TraceScreen();

            m_Handler.WaitForSession (SessionDisplayWaitType.PLUSLU, 5000);
            TraceScreen();

            m_Handler.WaitForContent(@"DEMONSTRATION", 20000);
            TraceScreen();

            // Enable clear screen.
            EnableClearScreen();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void ClearScreen_Click(object sender, EventArgs e)
    {
        try
        {
            // Disable every button and text box.
            DisableEverything();

            m_Handler.SendKey("@C");
            TraceScreen();

            m_Handler.WaitForSession(SessionDisplayWaitType.NotBusy, 5000);
            TraceScreen();

            // Enable enter ccli.
            EnableCCLI();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void SendCCLI_Click(object sender, EventArgs e)
    {
        try
```

```
        {
            // Disable every button and text box.
            DisableEverything();

            m_Handler.SendKey("CCLI@E");
            TraceScreen();

            m_Handler.WaitForContent("Call duration in milliseconds", 20000);
            TraceScreen();

            // Get the Jane Doe cursor Position.
            m_row = m_Handler.Cursor.Row;
            m_column = m_Handler.Cursor.Column;

            // Enable IP Address, Port and Perform Transaction.
            EnableIPInfo();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void FillIPAddress_Click(object sender, EventArgs e)
    {
        try
        {
            IPAddress.Text = IPAddress.Text.Trim();
            if (IPAddress.Text.Length == 0)
            {
                MessageBox.Show("You must fill out the IP Address");
                return;
            }

            // Tab to the correct place from First Field.
            m_Handler.Cursor.Row = m_row;
            m_Handler.Cursor.Column = m_column;
            m_Handler.SendKey("@T@T");
            TraceScreen();

            m_Handler.SendKey(IPAddress.Text);
            TraceScreen();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void FillPort_Click(object sender, EventArgs e)
    {
        try
        {
            PortNumber.Text = PortNumber.Text.Trim();
            if (PortNumber.Text.Length == 0)
            {
                MessageBox.Show("You must fill out the Port Number");
                return;
            }

            // Tab to the correct place from First Field.
            m_Handler.Cursor.Row = m_row;
            m_Handler.Cursor.Column = m_column;
            m_Handler.SendKey("@T@T@T");
            TraceScreen();

            m_Handler.SendKey(PortNumber.Text);
            TraceScreen();
```

```
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void PerformTX_Click(object sender, EventArgs e)
    {
        try
        {
            // Disable every button and text box.
            DisableEverything();

            m_Handler.SendKey("@E");
            TraceScreen();

            // Wait for screen to calm down.
            m_Handler.WaitForSession(SessionDisplayWaitType.NotBusy, 5000);
            TraceScreen();

            // See if the Balance Field is filled out.
            m_Handler.Cursor.Row = m_row;
            m_Handler.Cursor.Column = m_column;
            TraceScreen();
            // Tab to the Account Number field.
            m_Handler.SendKey("@T");
            TraceScreen();
            // Move to the Next Field (Empty Stuff after 123456).
            m_Handler.MoveNextField();
            TraceScreen();
            // Move to the Next Field (Title, Account Balance).
            m_Handler.MoveNextField();
            TraceScreen();
            // Move to the Next Field (Account Balance).
            m_Handler.MoveNextField();
            TraceScreen();

            // Extract Data from this field.
            string accountBalance = m_Handler.CurrentField.Data;

            // Trim the string.
            accountBalance = accountBalance.Trim();

            // Only things to do now are clear screen or disconnect.
            EnableClearScreen();

            // If we failed (not Abended) then this field will be blank.
            if (accountBalance.Length == 0)
                throw new Exception("Failed to get Account Balance");
            else
                MessageBox.Show(accountBalance, "Account Balance");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void Disconnect_Click(object sender, EventArgs e)
    {
        // Disable every button and text box.
        DisableEverything();

        m_Handler.Disconnect();

        // Go back to the original state of buttons.
        SetOpeningState();
```

```
        }

        // Get the Unicode version of the Screen.
        public String CurrentScreen()
        {
            if (m_Handler == null)
                throw new Exception("C3270_E_NOT_CONNECTED");

            String screen = null;

            ScreenData screenData = m_Handler.GetScreenData(1, 1, -1);

            // Convert the EBCDIC to Unicode.
            screen = HostStringConverter.ConvertEbcdicToUnicode(screenData.Data);

            return screen;
        }

        // Print out the 3270 screen to a provided text box.
        private void TraceScreen()
        {
            // If we are not connected, no info.
            if (m_Handler == null)
            {
                ScreenText.ResetText();
                return;
            }

            string screen = CurrentScreen();
            short rows = m_Handler.Rows;
            short columns = m_Handler.Columns;

            ScreenText.ResetText();
            for (int i = 0; i < rows; i++)
            {
                ScreenText.Text += (i != 0 ? Environment.NewLine : "") + screen.Substring(c
olumns * i, columns);
            }

            // Add a divider.
            ScreenText.Text += Environment.NewLine + new string('-', (int)columns);

            ScreenText.Refresh();
        }
    }
}
```

Using Session Integrator for TN3270

Session Integrator can be used to interact with TN3270 sessions. The behavior for Session Integrator using the TN3270 protocol is the same as Session Integrator for LU2 except for a different connection string.

Connection string for TN3270 sessions.

```
TRANSPORT=TN3270;TN3270SERVER="{IP address or
hostname}";TN3270PORT="{default=23}";DEVICETYPE="{default=IBM-3278-2}"
```

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Guide

This section discusses how to use the client-based BizTalk Adapter for WebSphere MQ.

In This Section

Correlating Messages using Request-Reply

See Also

Other Resources

Network Integration Programmer's Guide

Correlating Messages using Request-Reply

There are two ways to correlate messages in BizTalk Server orchestrations for MQSeries request-reply scenarios. The first is to supply the correlation identifier by setting both the MessageID (MQMD_MsgID) and the CorrelationID (MQMD_CorrelationID) to the same value. The second is to use the BizTalk_CorrelationId context property.

When sending the message to an MQSeries Queue Manager, you can set the message identifier (MQMD_MsgID) and the correlation identifier (MQMD_CorrelationID) to the same value in the outgoing message. The MQSeries Queue Manager copies the MessageID to the CorrelationID for the reply message. You can initialize the correlation sets for the outgoing message and follow the correlation sets for the incoming message using the value of MQMD_CorrelationID.

Alternately, instead of setting the MessageID and CorrelationID to the same value in the outgoing message, you can use the BizTalk_CorrelationID context property with a solicit-response send port of BizTalk Adapter for MQSeries.

To use identifiers provided by MQSeries Server for correlations in your BizTalk orchestration, BizTalk Server must first obtain the identifier. Your application does this through a solicit-response request. BizTalk Server sends a solicit-response request using MQSC Adapter to the MQSeries Server. In return, it receives a response with the message identifier (MQMD_MsgId) and the correlation identifier (MQMD_CorrelationId).

For the outgoing message in a solicit-response send port, the adapter copies the MQMD_MsgID generated by MQSeries to the MQSeries.BizTalk_CorrelationId context property.

When receiving messages, the adapter copies the MQMD_CorrelationId to the MQSeries.BizTalk_CorrelationId. In this case, using correlation sets, you can initialize the correlation sets for the outgoing message and follow the correlation sets for the incoming message using the MQSeries.BizTalk_CorrelationId.

Note
In the pre-release, the second mechanism is not supported.

See Also

Other Resources

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Guide

Administration and Management Programmer's Guide

This section describes how to use Windows Management Instrumentation (WMI) to programmatically manage different tasks for Host Integration Server.

In This Section

WMI and Host Integration Server

Using WMI with Host Integration Server

WMI and Host Integration Server

Windows Management Instrumentation (WMI) is a component of the Microsoft Windows operating system that provides management information and control in an enterprise environment. In the context of Microsoft Host Integration Server, WMI provides a programming and scripting interface for four major administrative tasks: configuration, run-time control, health monitoring, and trace capturing.

In This Section

WMI and the Host Integration Server Architecture

The architecture of the Windows Management Instrumentation (WMI) technology includes management applications, managed objects, providers, and the management infrastructure.

Management Applications

A management application is a Microsoft Windows-based application or service that processes or displays data from a managed object. A management application can perform a variety of tasks in a Host Integration Server 2009 environment, such as configuring servers running Host Integration Server, measuring performance, reporting outages, and correlating data. The management application is what you will probably be creating using this programmer's guide.

Managed Objects

A managed object represents a logical or physical enterprise component. A managed object is modeled in WMI using the Common Information Model (CIM), and is accessed by a management application through the WMI programming interface. A managed object in the Host Integration Server 2009 environment can be any component of the system, from a link service device driver communicating with hardware to software configuration information about users and connected logical units (LU).

WMI Providers

A WMI provider is a COM object that exposes an interface to a managed object. The WMI providers supplied with Host Integration Server 2009 use the WMI COM API to supply the WMI repository with data from Host Integration Server managed objects, to handle requests on behalf of Host Integration Server management applications, and to generate notifications of events.

Management Infrastructure

The management infrastructure is made up of WMI and the CIM repository. WMI lets users handle communications between management applications and providers. Users store their static data in the CIM repository. Applications and providers communicate through WMI using a common application programming interface (COM API). The COM API, which supplies event notification and query processing services, is available in the C and C++ programming languages.

The CIM repository holds static management data. Static data is data that does not regularly change. WMI also supports dynamic data, which is data that must be generated on demand because it changes frequently. Data can be placed in the CIM repository by WMI or network administrators. Information can be placed in the CIM repository using either the managed object format (MOF) language and the MOF Compiler or the WMI COM APIs. The WMI providers supplied with Host Integration Server 2009 use both mechanisms.

Management applications can access the COM API directly to interact with WMI and the CIM repository to make management requests of Host Integration Server. applications can also use other access methods such as Open Database Connectivity (ODBC) and HTML to make these requests. An ODBC driver for WMI is included with Windows 2000. The protocol used for communication between local and remote components is Distributed Component Object Model (DCOM).

See Also

Other Resources

WMI and Host Integration Server

What You Can Do to Administer Host Integration Server Using WMI

The Windows Management Instrumentation (WMI) providers for Host Integration Server are designed as programming and scripting administration interfaces. As such, these providers are optimized to perform get and set information, and to start and stop services. The following section describes what components of Host Integration Server you can administer with WMI, and what tasks you can perform on those components.

In This Section

Administrative Tasks That You Can Perform on Host Integration Server Using WMI

Using the Windows Management Instrumentation (WMI) providers for Host Integration Server, you can administer a variety of components on Host Integration Server. However, any administration you perform on Host Integration Server through WMI is performed through the following tasks:

- Logging on to WMI

  Like most providers, you must first log on to the WMI provider to use it. Therefore, the WMI providers expose the logon and security interfaces. For more information, see Logging on to Host Integration Server Through a WMI Provider.

- Getting and setting information

  Most of the tasks you will do through WMI will be checking information about different Host Integration Server services. You can also set information. For more information, see Accessing a Host Integration Server Property through WMI.

- Calling methods

  The methods exposed on the WMI providers for Host Integration Server are designed mainly to administer various Host Integration Server services. You can administer these services by calling the relevant **Start**, **Stop**, **Pause**, or **Cancel** method through the relevant provider. For more information, see
  Calling a Host Integration Server Method through WMI.

Sections of Host Integration Server That You Can Administer with WMI

Host Integration Server 2009 is included with Windows Management Instrumentation (WMI) providers that grant administrative access to the following areas:

- Rolling out, configuring, and restoring services

  The SNA Provider WMI Programmer's Reference and SNA Trace Provider WMI Programmer's Reference providers expose information regarding the Setup and rollout procedures for Host Integration Server 2009. You can use instances of classes such as **MsSna_LinkService** to set up and administer link services, and the classes derived from **MsSna_Config** to configure your system. For more information, see Configuring Host Integration Server with WMI.

- Starting and stopping services and connections

  The **WmiSna** provider exposes the **MsSna_Server**, **MsSna_Service**, and **MsSna_Connection** classes. These classes and their derived classes let you add an SNA service, start and stop services, pause services, and start and stop connections. For more information, see Controlling Services and Connections with WMI.

- Health monitoring

  The **WmiSnaStatus** provider exposes information regarding the SNA service status. **WmiSnaStatus** uses a variety of classes derived from **MsSnaStatus_Event** to describe changes in different parts of your system. For more information, see How to Monitor the Health of Host Integration Server with WMI.

- Trace capturing

  The **WmiSnaTrace** provider supports classes and instances derived from the **MsHisTrace_Config** class, which displays information regarding different configuration files. **WmiSnaTrace** also supports the **MsHisTrace_Event**, which describes an event that occurs whenever a trace is triggered. For more information, see How to Capture a Trace with WMI.

Additionally, Microsoft Windows 2000 includes several standard WMI providers, such as a registry provider, for accessing information from the system registry. Windows 2000 also supplies a Windows 2000 Event Log provider that enables applications to receive notifications of Windows 2000 and Host Integration Server 2009 events and to access the information stored in the Windows 2000 event log. Third-party vendors can create custom providers to interact with managed objects specific to their environment.

What You Should Know Before Programming for WMI and Host Integration Server

To use this guide effectively, you should be familiar with the following:

- Microsoft Host Integration Server 2009
- SNA concepts
- Windows Management Instrumentation (WMI)
- Microsoft Windows Server 2003 operating systems, Windows XP Professional, Windows 2000 Server

Depending on the API and development tools used, you should also be familiar with the following languages:

- WMI COM/DCOM APIs
- WMI Query Language
- WMI schema and MOF file syntax
- Windows Script Host
- Microsoft Visual Basic .NET
- Visual Basic for Applications
- Visual Basic Scripting Edition
- JScript
- Microsoft ODBC
- Microsoft ADO
- Active Server Pages

See Also
Other Resources
WMI and Host Integration Server

Supported Platforms for Administering with WMI

Microsoft Windows Management Instrumentation (WMI) is included as part of Microsoft Windows Server 2003, Windows XP, and Windows 2000. For use with Host Integration Server, the WMI Software Development Kit version 1.1 or later can be installed on one of the following operating systems:

- Windows Server 2003
- Windows XP Professional
- Windows 2000 Server
- Windows 2000 Advanced Server
- Windows 2000 Datacenter Server

Windows Script Host (WSH) is installed by default with Windows 2000.

Complete information about WSH is available as part of the Windows 2000 MSDN Platform SDK.

Using WMI with Host Integration Server

There are two main types of programming tasks described in this section: basic tasks and advanced tasks. A basic task is one of the tasks common to most every Windows Management Instrumentation (WMI) application: logging on to WMI, getting and setting properties, and calling methods. The advanced tasks are those built on the basic tasks, and describe the tasks for which the developers incorporated WMI into Host Integration Server. Additionally, this section discusses common problems that you may experience when installing and programming WMI for Host Integration Server, and solutions to those problems.

In This Section

Installing WMI on Host Integration Server

Host Integration Server 2009 should automatically install the relevant files for you to use the relevant Windows Management Instrumentation (WMI) interfaces. However, you may have to modify the location of certain managed object format (MOF) files if you want to use WMI on remote servers, and you may have to modify certain security settings if you want to have WMI work correctly as well.

In This Section

Installing WMI Providers on a Host Integration Server

Microsoft Host Integration Server 2009 provides several Windows Management Instrumentation (WMI) providers. The managed object format (MOF) files used by Host Integration Server are installed in the System directory below where Host Integration Server 2009 is installed. The default location for these files is in the following subdirectory:

C:\Program Files\Microsoft Host Integration Server\System

To use or view these MOF files on other computers (a computer with the Administration client or end-user client, for example) to develop applications, these MOF files must be copied from the Host Integration Server 2009 computer. These MOF files document the WMI providers supplied with Host Integration Server 2009 and the class identifiers (CLSIDs), classes, properties, and methods supported by these providers.

How to Upgrade to Windows Server 2003 with Host Integration Server and WMI

Windows Server 2003 interacts with Windows Management Instrumentation (WMI) providers differently than previous versions of the operating system. One change is how the operating system handles the credentials under which the provider is hosted. If you upgraded your Host Integration Server computer to Windows Server 2003, you may have to recompile the Host Integration Server WMI namespace using updated Windows Server 2003 MOF files. Recompiling with the new MOF files should let you use WMI on Windows Server 2003.

To recompile the Host Integration Server WMI namespace for Windows Server 2003

- Use the following syntax on the command prompt:

  Mofcomp -class:forceupdate mof filename

  The following examples show how to recompile each MOF file:

```
Mofcomp -class:forceupdate  wmiHIS_xp.mof
Mofcomp -class:forceupdate wmiHIS_xp.mof
Mofcomp -class:forceupdate wmisnastatus_xp.MOF
Mofcomp -class:forceupdate wmisnatrace_xp.mof
Mofcomp -class:forceupdate wmisna_xp.mof
```

How to Set ASP Security for Host Integration Server and WMI

WMI scripting using Active Server Pages (ASP) is enabled automatically on Microsoft Windows 2000. For the correct security setting for ASP on Windows 2000, it is recommended that you set **Anonymous Authentication** to **Off** and enable **Integrated Windows Authentication** in the Internet Information Services (IIS) configuration for directories with ASP files used with Host Integration Server. To access Host Integration Server configuration and status information, an application or user must have the appropriate administrative rights, which are not available with anonymous authentication. Use the following procedure on Windows 2000 to correctly configure security using ASP.

To configure security using ASP

1. To open IIS, click **Start**, point to **Administrative Tools**, point to **Services**, and then click **IIS Admin Service**, or click **Start**, point to **Settings**, click **Control Panel**, click **Administrative Tools**, click **Services**, and then click **IIS Admin Service**).
2. Move to the directory where the ASP files reside.
3. Right-click the directory, and then click **Properties**.
4. When the next dialog box appears, on the **Directory Security** tab, in the **Anonymous Authentication** section, click **Edit**.
5. When the next dialog box appears, clear the **Anonymous Authentication** check box, select **Integrated Windows Authentication**, and then click **OK** to save these settings.

This sets that particular directory to use **Integrated Windows Authentication** instead of **Anonymous Authentication** without affecting any of your other directories. If there are other ASP files that require or allow **Anonymous Authentication** you may want to create a new directory in which you can turn off **Anonymous Authentication** and store the WMI ASPs there. Any script that calls **ExecMethod** from an ASP page should be set up to use **Integrated Windows Authentication** to verify the user trying to run the script.

Additionally, when using a "REFRESH" variable on a Web page and the page is being used to start and stop SNA service through ASP scripting, the Web browser client (Internet Explorer, for example) should set the **Every visit to the page** option, as shown in the following procedure.

To use a Web browser client to start and stop SNA services through ASP scripting

1. Click **Start**, point to **Programs**, and then click **Internet Explorer**.
2. In Internet Explorer, on the **Tools** menu, click **Internet Options**.
3. In the **Internet Options** dialog box, on the **General** tab, in the **Temporary Internet file** section, click **Settings**.
4. In the **Settings** dialog box, in the **Check for newer versions of stored pages** section, make sure that the option **Every visit to the page** is selected.
5. Click **OK**.

If this change is not made on the Web browser client, some ASP scripts do not run correctly because Internet Explorer is caching older results.

Basic WMI Tasks for Host Integration Server

Regardless of what you may want to do with Windows Management Instrumentation (WMI), most actions you will perform with it are one of the following: logging on to WMI, getting or setting WMI properties, or calling WMI methods.

In This Section

Logging on to Host Integration Server Through a WMI Provider

The first step that you must perform when you create a WMI application or script is to log on to WMI and set the security for your application. You can perform this action either by using the **SWbemLocator** locator object, or with a moniker.

To connect to WMI using SWbemLocator

1. Retrieve a locator object with a call to **CreateObject**.
2. Log on to the namespace with a call to **ConnectServer**.
3. Set the impersonation level with a call to **Security._ImpersonationLevel**.
4. Implement your task.

The following code sample shows how to connect to WMI using **SWbemLocator**:

```
    Set WmiLocator = CreateObject("WbemScripting.SWbemLocator")
    Set WmiNameSpace = WmiLocator.ConnectServer("","root\MicrosoftHIS","", "","", "",0,Noth
ing)

    if Err = 0 then
        'Retrieve the SNA_LU_Lua class
        Set ServerClass = WmiNamespace.Get("MsSNA_LuLua")
        Set Path = ServerClass.Path_
        ServerClass.Security_.impersonationLevel = 3
        Set LU3270 = ServerClass.Instances_
```

Another way you can connect to WMI is by using a moniker. A moniker is essentially a compact version of the above lines of code, and contains the WMI namespace and other connection information.

To connect to WMI using a moniker

1. Call **GetObject** with a moniker in the input parameter.
2. Implement your task.

The following example shows how to connect to WMI using a moniker:

```
    set objService = GetObject("winmgmts:root/microsofthis")
```

Accessing a Host Integration Server Property through WMI

Getting and setting an instance is one of the most common retrieval procedures you are most likely to perform on Host Integration Server using Windows Management Instrumentation. You can retrieve an instance using the **GetObject** method, and modify it using the various **Put_** methods.

In This Section

How to Retrieve an Instance

You can retrieve a local copy of the instance with a call to **GetObject** with the object path of the instance. The following example shows how to retrieve an IPDLC connection using WMI:

```
Set ObjClass = Namespace.Get("MsSna_ConnectionIpDlc")
```

Besides retrieving a single instance, you can also retrieve collections of WMI instances by querying the WMI database with a **SELECT** statement.

How to Retrieve Multiple Instances

You can retrieve a local copy of the collection with a call to **ExecQuery**. After you have retrieved an instance, you can modify and update the instance. The following example shows how to retrieve a collection:

```
Set colItems = objWMIService.ExecQuery("Select * from MSSnaStatus_Connection")
```

How to Modify or Update an Instance

After you have retrieved an instance, you can modify your local copy and update your changes to the server.

To modify or update an instance

1. Retrieve a local copy of the object with a call to **GetObject**.
2. If necessary, view the properties of the object with a call to the **Properties_** method.

   Although not required, you may want to know the value of the property before you change it.
3. Make any changes to the object properties with a call to the **SWbemProperty.Value** method.

   The **Value** method changes only the local copy. To save your changes to WMI, you must put the complete copy back in the WMI repository.
4. Put the object back in the WMI repository with a call to the **SWbemObject.Put_** or **SWbemObject.PutAsync_** methods.

   As the names imply, **Put_** updates synchronously while **PutAsync_** updates asynchronously. Either method copies over the original instance with your modified instance. However, to take advantage of asynchronous processing, you must create a **SWbemSink** object.

   The following example shows how to update an instance:

```
    Set ObjClass  = Namespace.Get("MsSna_LinkService_IpDlc")
' Create new link service instance
    Set NewInst   = ObjClass.SpawnInstance_
    ' Set instance properties
    NewInst.NetworkName = Left(strComputerName, 8)
    NewInst.CPName = "IPDLCLS"
    NewInst.NodeID = "05D.FFFFF"
    NewInst.AddressType = 2
    NewInst.LocalAddress = Trim(strLocalAddress)
    NewInst.LENNode = strLenNode
    NewInst.PrimaryNNS = strPrimaryNNS
    if (strBackupNNS <> Empty) then
        NewInst.BackupNNS = strBackupNNS
    end if
    ' Commit the instance
    NewInst.Put_
```

Calling a Host Integration Server Method through WMI

You have two options when calling methods through WMI: you may call a WMI method, or a provider method. A WMI method is supported by the WMI infrastructure, and provides general services such as query support or log on access. For example, Logging on to Host Integration Server Through a WMI Provider and Accessing a Host Integration Server Property through WMI describe calling the **GetObject** and **ExecQuery** WMI methods. In contrast, a provider method is supported by the provider, and is unique to each service. Most HIS provider methods deal with controlling services. For example, the **WmiSna** provider supports the **MsSna_ServiceSNA** class, which in turn supports the **Start**, **Stop**, **Pause**, and **Resume** methods. The actual process of calling a WMI method or a provider method is the same as calling any other COM or scripting interface.

Advanced WMI Tasks for Host Integration Server

An advanced Windows Management Instrumentation (WMI) task is a task that uses the basic WMI programming tasks to perform an activity specific to Host Integration Server. Generally, an advanced task deals with setting up Host Integration Server on a server, controlling a service or connection, or monitoring an aspect of your enterprise.

In This Section

Configuring Host Integration Server with WMI

One of the reasons Windows Management Instrumentation (WMI) is implemented in Host Integration Server is to help in configuring and updating different technologies in your enterprise. As an extended sample, this section describes how to configure an IPDLC link service. This sample is a command prompt script that takes as input the host you want to go against and configures an IPDLC link service and independent session connection to the host.

In This Section

How to Configure an IPDLC Link Service

To configure an IPDLC link service

1. Retrieve the command prompt instructions.
2. If necessary, retrieve the adapter name.
3. Create the link service.
4. Create the independent session.

The following code sample describes how to configure an IPDLC link service. The following table describes the relevant command prompt switches:

Command line switch	Description
-p	PrimaryNNS: the Primary Network Node Server
-b	(Optional) BackupNNS: Backup Network Node Server
-d	(Optional) device: Overrides the adapter name discovered
-l	(Optional) LenNode: Used to configure the LEN node (default is 1st)

```
    On Error Resume   Next
'Declarations and Constants
    Dim strPrimaryNNS
    Dim strBackupNNS
    Dim strLocalAddress
    Dim strLenNode
    Dim strComputerName
    Dim wshEnvProc
'Defaults
    strPrimaryNNS = ""
    strBackupNNS = ""
    strLocalAddress = ""
    strLenNode = "SNA Service"
'Get the command line
    Set objArgs = WScript.Arguments
    For I = 0 to objArgs.Count - 1
       select case objArgs(I)
        Case "-p"
            I = I + 1
            strPrimaryNNS = objArgs(I)
        case "-b"
            I = I + 1
            strBackupNNS = objArgs(I)
        case "-d"
            I = I + 1
            strLocalAddress = objArgs(I)
        case "-l"
            I = I + 1
            strLenNode = objArgs(I)
        case else
            Wscript.Echo "Script Usage: "
            Wscript.Echo " -p Primary NNS (Required)"
            Wscript.Echo " -b Backup NNS (Optional)"
            wscript.echo " -d Device Name (Optional)"
            wscript.echo " -l Len Node (Optional)"
            return
       end select
    Next
'NNS parameter is required
    if strPrimaryNNS = "" then
```

```
        Wscript.Echo "Please supply the -p (PrimaryNNS) parameter"
        return
    end if
'See if we need to get a device
    if strLocalAddress = "" then
        GetAdapterName()
    end if
' Two methods for creating the link service and independent session connection
    CreateIPDLCLinkService
    CreateIndependentSession
```

How to Retrieve an Adapter Name

One of the tasks you must perform when setting up an IPDLC connection is to retrieve the name of the adapter you are connecting with.

To retrieve an adapter name

1. Connect to the namespace on the local computer using **GetObject**.
2. Retrieve the name of the adapter using **ExecMethod** with **GetAllNetworkAdapters** as the method to execute.

The following example shows how to retrieve the name of the first adapter on a system:

```
Private Sub GetAdapterName()
   Dim objService, outParam, objSD, MyArray, nArray
   set objService = GetObject("winmgmts:root/microsofthis")
   set outParam = objService.Execmethod("MsSna_LinkService_IPDLC",
"GetAllNetworkAdapters")
   objSD = Join(outParam.Adapters, ",")
   MyArray = Split(objSD, ",")
   nArray = Ubound(MyArray)
   if nArray < 0 then
      strLocalAddress = ""
   else
      strLocalAddress = MyArray(0)  'default to first one
   end if
End Sub
```

How to Create a Link Service

Another task you may want to perform when setting up an IPDLC connection is to create the link service.

To create a link service

1. Connect to the namespace on the local computer using **GetObject**.
2. Create the new link service instance using **SpawnInstance**.
3. Set the properties of the new link service.
4. Commit the new instance to memory using the **Put_** method.

The following example shows how to create a new link service:

```
private Sub CreateIPDLCLinkService
    on error resume next
' Connect to the namepsace on the local machine
    Set Namespace = GetObject("Winmgmts:root\MicrosoftHIS")
    Set ObjClass  = Namespace.Get("MsSna_LinkService_IpDlc")
' Create new link service instance
    Set NewInst   = ObjClass.SpawnInstance_
    ' Set instance properties
    NewInst.NetworkName = Left(strComputerName, 8)
    NewInst.CPName = "IPDLCLS"
    NewInst.NodeID = "05D.FFFFF"
    NewInst.AddressType = 2
    NewInst.LocalAddress = Trim(strLocalAddress)
    NewInst.LENNode = strLenNode
    NewInst.PrimaryNNS = strPrimaryNNS
    if (strBackupNNS <> Empty) then
        NewInst.BackupNNS = strBackupNNS
    end if
    ' Commit the instance
    NewInst.Put_
    if Err.Number <> 0 then
        PrintWMIErrorThenExit Err.Description, Err.Number
        Wscript.Echo "Link Service Creation Failed " & Err.Description
    Else
        Wscript.Echo "Link Serice Created Successfully"
    end if
End Sub
```

How to Create an Independent Session

After you have created the link service, you need to create an independent session to use.

To create an independent session

1. Create the "Independent Sessions Connection" record.
2. Validate that the session was created.
3. Define the properties of the new connection.

The following code sample shows how to create an independent session:

```
 Private Sub CreateIndependentSession
     On error resume next
 'Create the Independent Sessions Connection record.
     Set Namespace = GetObject("Winmgmts:root\MicrosoftHIS")
 'Validate that the instance was created
     strQuery = "select * from MsSna_LinkService_IpDlc"
     ' this is our instance
     Set instset = Namespace.ExecQuery(strQuery)
     if ( instset.Count<>1) Then
         wscript.echo "No instances found for the link service query " & strQuery
     End If
     For each inst_ in instSet
         Set Inst = inst_ ' This is our new instance
     Next ' end of query workaround
     ' define independent sessions connection
     Set ObjClass = Namespace.Get("MsSna_ConnectionIpDlc")
     Set IndepConn = ObjClass.SpawnInstance_
     IndepConn.Activation = 0
     IndepConn.Adapter = "SNAIP1"
     IndepConn.AllowIncoming = TRUE
     IndepConn.BackupDLUSCPName = ""
     IndepConn.BackupDLUSNetName = ""
     IndepConn.BlockId = "05D"
     IndepConn.Comment = ""
     IndepConn.CompressionLevel = 0
     IndepConn.DLURRetryDelay = 0
     IndepConn.DLURRetryLimit = 0
     IndepConn.DLURRetryType = 0
     IndepConn.DynamicLuDef = TRUE
     IndepConn.IndepSess = TRUE
     IndepConn.LocalControlPoint = ""
     IndepConn.LocalNetName = ""
     IndepConn.Name = Inst.Name
     IndepConn.NodeId = "FFFFF"
     IndepConn.PartnerConnectionName = ""
     IndepConn.PeerRole = 1
     IndepConn.PrimDLUSCPName = ""
     IndepConn.PrimDLUSNetName = ""
     IndepConn.RemoteAddress = ""
     IndepConn.RemoteBlockId = ""
     IndepConn.RemoteControlPoint = ""
     IndepConn.RemoteEnd = 1
     IndepConn.RemoteNetName = ""
     IndepConn.RemoteNodeId = ""
     IndepConn.RetryDelay = 0
     IndepConn.RetryLimit = 0
     IndepConn.Service = strComputerName
     IndepConn.XIDFormat = 1
     IndepConn.Put_
     if Err.Number <> 0 then
         PrintWMIErrorThenExit Err.Description, Err.Number
     End If
 End Sub
```

See Also

Reference

Creating a Link Service

How to Handle Errors While Creating a Link Service

As with most scripts, you need to write functions to handle any errors. This error handler scans the Windows Management Instrumentation (WMI) error queue for any relevant error information and posts and displays the error to the user.

The following sample shows how to handle errors while you are creating a link service:

Syntax

```
Sub PrintWMIErrorThenExit(strErrDesc, ErrNum)
    On Error Resume Next
    Dim objWMIError : Set objWMIError =    CreateObject("WbemScripting.SwbemLastError")
    wscript.echo TypeName(objWMIError)

    If ( TypeName(objWMIError) = "Empty" ) Then
    wscript.echo strErrDesc & " (HRESULT: " & Hex(ErrNum) & ")."
    Else
    wscript.echo "Extended error information:"
    wscript.echo objWMIError.Description
    Set objWMIError = nothing
    End If
    Exit sub
End Sub
```

Controlling Services and Connections with WMI

Another task that Windows Management Instrumentation (WMI) can perform is accessing services and connections throughout Host Integration Server. Two common procedures in this realm are displaying the status of all current connections, and retrieving information about a specified connection.

In This Section

How to Display Connection Status

Retrieving the status of a connection is a common task that you might want to perform with WMI.

To display the status of a connection

1. Connect to the namespace using **GetObject** with a moniker in the parameter.
2. Enumerate **MsSnaStatus_Connections** using **ExecQuery**.
3. Display error codes if necessary.

The following example shows how to display the status of all the connections defined in Host Integration Server (HIS):

```
Private Function DisplayConnectionStatus ()
'Variables
   Dim objWMIService, colItems, iCounter, objItem, _
       strReport
'Connect to the namespace
   Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\microsofthis")
'Enumerate the Class
   Set colItems = objWMIService.ExecQuery("Select * from MSSnaStatus_Connection")
   iCounter = colItems.Count
   if Err.Number = 0 then
      For Each objItem in colItems
         strReport = "Connection " & objItem.Name & " status is " & objItem.StatusText
         Wscript.Echo strReport
         strReport = ""
      Next
   else
      Wscript.Echo "An error occurred enumerating instances for status " & Err.Number & " "
& Err.Description
   End If
   DisplayConnectionStatus = true
End Function
```

How to Retrieve Connection Information

Another common task you may want to perform with Windows Management Instrumentation (WMI) for Host Integration Server is retrieving information regarding a connection.

To retrieve connection information

1. Connect to the namespace using **GetObject** with a moniker in the parameter.
2. Retrieve all connection information objects using **ExecQuery**.
3. Display the information as appropriate.

The following code sample shows how to retrieve information about an SDLC connection:

```
On Error Resume Next
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\MicrosoftHIS")
Set colItems = objWMIService.ExecQuery("Select * from MsSna_ConnectionSdlc",,48)
For Each objItem in colItems
    Wscript.Echo "Activation: " & objItem.Activation
    Wscript.Echo "Adapter: " & objItem.Adapter
    Wscript.Echo "AllowIncoming: " & objItem.AllowIncoming
    Wscript.Echo "BlockId: " & objItem.BlockId
    Wscript.Echo "Comment: " & objItem.Comment
    Wscript.Echo "CompressionLevel: " & objItem.CompressionLevel
    Wscript.Echo "DialData: " & objItem.DialData
    Wscript.Echo "DynamicLuDef: " & objItem.DynamicLuDef
    Wscript.Echo "LocalControlPoint: " & objItem.LocalControlPoint
    Wscript.Echo "LocalNetName: " & objItem.LocalNetName
    Wscript.Echo "MaxBtu: " & objItem.MaxBtu
    Wscript.Echo "Name: " & objItem.Name
    Wscript.Echo "NodeId: " & objItem.NodeId
    Wscript.Echo "PartnerConnectionName: " & objItem.PartnerConnectionName
    Wscript.Echo "PeerRole: " & objItem.PeerRole
    Wscript.Echo "RemoteBlockId: " & objItem.RemoteBlockId
    Wscript.Echo "RemoteControlPoint: " & objItem.RemoteControlPoint
    Wscript.Echo "RemoteEnd: " & objItem.RemoteEnd
    Wscript.Echo "RemoteNetName: " & objItem.RemoteNetName
    Wscript.Echo "RemoteNodeId: " & objItem.RemoteNodeId
    Wscript.Echo "RetryDelay: " & objItem.RetryDelay
    Wscript.Echo "RetryLimit: " & objItem.RetryLimit
    Wscript.Echo "SdlcContactLimit: " & objItem.SdlcContactLimit
    Wscript.Echo "SdlcContactTO: " & objItem.SdlcContactTO
    Wscript.Echo "SdlcDataRate: " & objItem.SdlcDataRate
    Wscript.Echo "SdlcDuplex: " & objItem.SdlcDuplex
    Wscript.Echo "SdlcEncoding: " & objItem.SdlcEncoding
    Wscript.Echo "SdlcIdleLimit: " & objItem.SdlcIdleLimit
    Wscript.Echo "SdlcIdleTO: " & objItem.SdlcIdleTO
    Wscript.Echo "SdlcLeasedLine: " & objItem.SdlcLeasedLine
    Wscript.Echo "SdlcMultiPrimary: " & objItem.SdlcMultiPrimary
    Wscript.Echo "SdlcPollAddress: " & objItem.SdlcPollAddress
    Wscript.Echo "SdlcPollLimit: " & objItem.SdlcPollLimit
    Wscript.Echo "SdlcPollRate: " & objItem.SdlcPollRate
    Wscript.Echo "SdlcPollTO: " & objItem.SdlcPollTO
    Wscript.Echo "SdlcStandby: " & objItem.SdlcStandby
    Wscript.Echo "SdlcSwitchTO: " & objItem.SdlcSwitchTO
    Wscript.Echo "Service: " & objItem.Service
    Wscript.Echo "StatusText: " & objItem.StatusText
    Wscript.Echo "XIDFormat: " & objItem.XIDFormat
Next
```

How to Monitor the Health of Host Integration Server with WMI

Health monitoring refers to viewing the current status of different aspects of Host Integration Server through the **WmiSnaStatus** provider. You can access **WmiSnaStatus** provider in the same manner as you would any other instance and method provider. You use **ExecQuery** to query for the relevant information, and then display the information to screen or write the information to a log file. Some **WmiSnaStatus** classes also have methods that let you start and stop related services.

To monitor the health of Host Integration Server with WMI

1. Connect to the namespace using **GetObject** with a moniker in the parameter.
2. Retrieve the object representing the SNA Status provider using **ExecQuery**.
3. Use the information gathered from the **ExecQuery** as appropriate.

You can see an example of health monitoring in How to Display Connection Status.

How to Capture a Trace with WMI

Trace capturing refers to the process of viewing trace logs. For Host Integration Server, this typically refers to collating data stored in trace log objects into a single file and saving it to a specified location. You can capture SNA trace information in the same manner as you would retrieve any other information from Windows Management Instrumentation (WMI). You use **ExecQuery** to make a call to the relevant object, and then write the information to the location you want.

To capture a trace

1. Connect to the namespace using **GetObject** with a moniker in the parameter.
2. Retrieve the objects representing the SNA Application using **ExecQuery**.

   The core functionality of capturing a trace can be described in the following code:

   ```
   Set colItems = objWMIService.ExecQuery("Select * from MsHisTrace_SNAApplication",,48)
   Set colItems = objWMIService.ExecQuery("Select * from MsHisTrace_SNABase",,48)
   ```

   Everything else in this sample is to support logging to a file.

The following code example shows how to capture a trace:

```
On Error Resume Next
strComputer = "."
Dim iCounter
'Initialize
    CreateLogFile
    Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\MicrosoftHIS")

'Validate TraceSnaApplication
    iCounter = 0
    Set colItems = objWMIService.ExecQuery("Select * from MsHisTrace_SNAApplication",,48)
For Each objItem in colItems
    Wscript.Echo "APPCTrace: " & objItem.APPCTrace
    Wscript.Echo "CPICTrace: " & objItem.CPICTrace
    Wscript.Echo "CSVTrace: " & objItem.CSVTrace
    Wscript.Echo "EnabledTraces: " & objItem.EnabledTraces
    Wscript.Echo "InternalMessageTrace: " & objItem.InternalMessageTrace
    Wscript.Echo "LU62Trace: " & objItem.LU62Trace
    Wscript.Echo "LUATrace: " & objItem.LUATrace
    Wscript.Echo "T3270Trace: " & objItem.T3270Trace
    iCounter = iCounter + 1
Next

if iCounter > 0 then
    Wscript.Echo "Number of Instances found " & iCounter
else
    Wscript.Echo "No Instances Found"
End If

    iCounter = 0
    Set colItems = objWMIService.ExecQuery("Select * from MsHisTrace_SNABase",,48)
For Each objItem in colItems
    Wscript.Echo "EnabledTraces: " & objItem.EnabledTraces
    Wscript.Echo "InternalMessageTrace: " & objItem.InternalMessageTrace
    Wscript.Echo "LU62Trace: " & objItem.LU62Trace
    Wscript.Echo "T3270Trace: " & objItem.T3270Trace
    iCounter = iCounter + 1
Next

if iCounter > 0 then
    Wscript.Echo "Number of Instances found " & iCounter
else
    Wscript.Echo "No Instances Found"
End If
```

Programming Considerations When Using WMI with Host Integration Server

The following section discusses various issues that you need to know about when programming Windows Management Instrumentation (WMI) with Host Integration Server.

In This Section

Using Host Integration Server and WMI on a Backup Server

There are some restrictions on using Windows Management Instrumentation (WMI) and Host Integration Server 2009 regarding connections to backup servers. The Snabase works to synchronize the information in the COM.CFG configuration file on the primary server across all backup servers. Each backup server has a local copy of the COM.CFG file from this synchronization process. WMI has a limitation that it will not attempt to read the local backup server's copy of COM.CFG if the primary server is alive. This request will always be forwarded to the primary server.

A client that connects to a WMI provider running on a backup server cannot retrieve any information or make configuration changes. This is a limitation of DCOM which does not permit impersonation outside the local computer. When a client on one computer connects to a WMISNA provider on another computer that is a backup server, the client is using DCOM to connect to the backup Host Integration Server computer. When the WMI provider on the backup server tries to access the COM.CFG file from the primary server, this is not allowed by DCOM because the application is trying to impersonate the user across the computer boundary.

You can work around this limitation on Windows 2000 using delegation.

The limitation against accessing a remote COM.CFG file does not apply when the primary server is down. In this case the server will fail-over to a backup and use replicated copies.

Using Duplicate LU Pools with Host Integration Server and WMI

A VBScript ImportExport sample program written in Microsoft Visual Basic Scripting Edition (VBScript) is provided as part of the Host Integration Server SDK. This tool enables configuration information from Host Integration Server to be exported and saved to a text file using Windows Management Instrumentation (WMI) in MOF format. This text file can also be changed and imported using this sample program to change configuration information.

A potential problem using WMI can occur with duplicate LU pools that can be illustrated using this sample program. Typically, exporting and re-importing the MOF file would not create duplicates. However, the Host Integration Server WMI provider allows pool-to-workstation association instances to be duplicated because, by design, duplicates of this type of object are allowed. You can associate the same pool to the same workstation or user multiple times. This is used by emulators to create more sessions for clients. Therefore, you cannot identify one such association from another. The WMISNA provider, WMISNA.DLL, always creates new associations of these types, even if an association with the same pair (Pool, Wks) already exists. This object type is allowed only in this specific case. However, this can create a problem for applications developed using WMI (the Import/Export sample, for example) if the application does not know not to create the duplicates.

The follow sequence illustrates this issue using the ImportExport sample:

1. Use SNA Manager to create a pool workstation association.
2. Export the SNA configuration to a MOF file using the ImportExport utility.
3. Import that same MOF file again using the ImportExport utility.
4. Duplicate pool-workstation associations are created.

The result is that if a client uses the import/export sample or a similar application developed using WMI on a Host Integration Server configuration that has pool-to-workstation associations, then the number of associations will effectively double after running the sample. The workaround using the ImportExport sample would be as follows:

1. Export the configuration to a MOF file.
2. Remove the pool to workstation associations from the MOF file that was just created.
3. Re-import the MOF file.

When importing the configuration from one domain to another using the ImportExport sample or a similar application developed using WMI, then step 2 should be ignored. Typically, WMI applications should copy an existing configuration to a blank configuration file so this condition does not occur.

Messaging Programmer's Guide

This section of the Host Integration Server 2009 SDK describes the extensions and components that make up Microsoft MSMQ-MQSeries Bridge.

For general information about developing for MSMQ-MQSeries Bridge, see Messaging Programmer's Reference.

For sample programs illustratating MSMQ-MQSeries Bridge, see Messaging Samples.

In This Section

MSMQ-MQSeries Bridge Programmer's Guide

MSMQ-MQSeries Bridge Programmer's Guide

The MSMQ-MQSeries Bridge provides the ability to send and receive messages between Message Queuing (also known as MSMQ) and IBM MQSeries easily and efficiently. The MSMQ-MQSeries Bridge Extensions enable a programmer to develop and control how these message transfers will occur and how properties of a message are translated.

The main programming issues when using the MSMQ-MQSeries Bridge Extensions fall into three areas:

- Queue addressing
- Message conversion
- Limitations to specific API functions

Queue addressing deals with how to specify the name of an MQSeries destination queue in a Message Queuing API call, or the name of an MSMQ destination queue in an MQSeries API call. Message conversion deals with how the MSMQ-MQSeries Bridge converts MSMQ message properties to MQSeries message data structures, and how the MSMQ-MQSeries Bridge converts MQSeries message data structures to MSMQ properties. These topics are covered in detail in separate sections. Limitations to the MSMQ-MQSeries Bridge Extensions are discussed in detail in the section on programming considerations.

In This Section

Platforms Supported by MSMQ-MQSeries Bridge Extensions

The MSMQ-MQSeries Bridge Extensions can access message queues on IBM MQSeries systems through SNA LU 6.2 or TCP/IP using Host Integration Server 2009 and MSMQ-MQSeries Bridge:

On Windows 2000, The MSMQ-MQSeries Bridge requires that Message Queuing (also known as MSMQ) be set up as a Message Queuing server, not a workgroup, with routing enabled. MSMQ-MQSeries Bridge Manager, which is used to configure and manage MSMQ-MQSeries Bridge, can be installed on any of the platforms supported by Host Integration Server 2009.

Queue Addressing Using MSMQ-MQSeries Bridge

This section explains how to specify the name of an MQSeries destination queue in a Message Queuing (also known as MSMQ) call, or the name of a Message Queuing destination queue in an MQSeries call. This information is needed to specify the destination queue where you are sending a message.

In This Section

Addressing an MQSeries Queue in Message Queuing

In Message Queuing, address MQSeries queues as if they are on the Message Queuing network. For the Message Queuing computer name, specify the MQSeries Queue Manager name. For the Message Queuing queue name, specify the MQSeries queue name.

For example, if the MQSeries Queue Manager is MQS1, and the queue name is **MQS_QUEUE4**, the Message Queuing path name is **MQS1\MQS_QUEUE4**.

Sending a Message Queuing Message to an MQSeries Queue

To send a message to an MQSeries queue, you follow the normal Message Queuing (also known as MSMQ) procedure, which is to determine the Message Queuing format name corresponding to the path name. More precisely, you must determine the format name of the Message Queuing foreign queue representing the MQSeries queue. If the foreign queue does not already exist, you can create it and determine its format name by calling **MQCreateQueue**. If the foreign queue already exists, you can call **MQPathNameToFormatName**, or you can determine the format name in Message Queuing Explorer.

Call **MQOpenQueue** with the format name argument to open the queue for send access.

Call **MQSendMessage** and specify the destination queue handle returned by **MQOpenQueue**.

In the MSMQ-to-MQSeries direction, MSMQ-MQSeries Bridge sends transacted messages using the MSMQ to MQS message pipe and untransacted messages using the MSMQ to MQS transactional message pipe.

Addressing a Message Queuing Queue in MQSeries

There are two ways to address a Message Queuing (also known as MSMQ) queue from MQSeries:

- By the Message Queuing format name
- By the Message Queuing path name

By either method, you specify the name in the object descriptor (MQOD structure) of the MQSeries message.

Note

MSMQ-MQSeries Bridge supports the following types of format names:

```
PUBLIC=<GUID>
PRIVATE=<machine GUID>\<file number>
DIRECT=OS:<Path name>
```

Note

For detailed addressing syntax and examples, see the section on Object Descriptors under Converting Messages Sent from MQSeries to Message Queuing.

Sending a Message to a Message Queuing Queue in MQSeries

To send a message from MQSeries to Message Queuing (also known as MSMQ), you specify the Message Queuing format or path name of the destination queue in the object descriptor. Call **MQOPEN** to open the queue. Call **MQPUT** or **MQPUT1** to send the message.

The addressing syntax enables you to send a message by either MQS->Message Queuing message pipe or MQS->Message Queuing transactional message pipe.

Converting Messages Using MSMQ-MQSeries Bridge

This section describes how the MSMQ-MQSeries Bridge converts Message Queuing (also known as MSMQ) message properties to MQSeries message data structures and how MQSeries data structures are converted to Message Queuing message properties.

When the MSMQ-MQSeries Bridge transmits a message, it converts the message properties between the Message Queuing and MQSeries formats. When equivalent properties exist in the two systems, the MSMQ-MQSeries Bridge assigns the property values directly. For example, the Message Queuing **messagebody** property is converted to the MQSeries **messagebuffer**. The Message Queuing **messagebodylength** is converted to the MQSeries **messagebufferlength**.

When partially equivalent properties exist in the two systems, the MSMQ-MQSeries Bridge assigns the properties according to conversion rules. For example, the MQSeries property **MQMD.Report** is converted to the Message Queuing properties **PROPID_M_ACKNOWLEDGE** and **PROPID_M_JOURNAL**.

When a property has no equivalent, the MSMQ-MQSeries Bridge either ignores the property or assigns a default value. For example, the Message Queuing property **PROPID_M_AUTH_LEVEL** refers to a specific Message Queuing authentication method that is not supported by MQSeries. In a message sent from Message Queuing to MQSeries, this property is ignored. In a message received by Message Queuing from MQSeries, the MSMQ-MQSeries Bridge assigns the Message Queuing default value to the property.

You can supplement or override the conversions described here using the Message Queuing message extension property (**PROPID_M_EXTENSION**). For information about overriding the default conversions, see Using Message Extensions.

When you send a message from Message Queuing to IBM MQSeries, MSMQ-MQSeries Bridge converts the Message Queuing message properties to an MQSeries data structure. To do this, MSMQ-MQSeries Bridge maps the various message properties of the Message Queuing message as nearly as possible to equivalent MQSeries fields. The following sections describe the conversion rules by which this is done.

In This Section

Converting Messages Sent from Message Queuing to MQSeries

The information in this section applies to messages that you send from Message Queuing (also known as MSMQ) to MQSeries. For messages sent from MQSeries to Message Queuing, see Converting Messages Sent from MQSeries to Message Queuing.

This section contains two main subsections, which provide essentially the same information but from complementary points of view.

The first section describes the conversion rules from the sender's point of view, and explains how MSMQ-MQSeries Bridge converts each Message Queuing property that you include in a message.

The second section explains the rules from the receiver's point of view. This section describes how MSMQ-MQSeries Bridge builds a complete MQSeries message containing all the needed fields, whether or not they have exact Message Queuing equivalents.

You can supplement or override the conversions described in this section by using the Message Queuing message extension property (**PROPID_M_EXTENSION**).

In This Section

- Converting Message Queuing Properties
- Building an MQSeries Message

Converting Message Queuing Properties

This topic explains how MSMQ-MQSeries Bridge converts the Message Queuing (also known as MSMQ) properties that you include in a message to MQSeries. For information about MQSeries fields that have no Message Queuing equivalents, see Building an MQSeries Message.

In This Section

Message Body (PROPID_M_BODY)

The Message Queuing (also known as MSMQ) message body is equivalent to the MQPUT or MQGET message buffer of MQSeries. The length of the MQPUT buffer is the Message Queuing message body size.

The following table lists the Message Queuing properties and the MQSeries fields to which they are converted.

Message Queuing property	Converted to MQSeries field
PROPID_M_BODY	Message buffer
PROPID_M_BODY_SIZE	Message buffer length

Queue Format Names (PROPID_M_..._QUEUE)

Each Message Queuing (also known as MSMQ) message contains the format name of a destination queue, and optionally the format names of response or administration queues. The MSMQ-MQSeries Bridge converts the names to the equivalent MQSeries object and Queue Manager names.

If a message contains both a response queue and an administration queue, MSMQ-MQSeries Bridge ignores the administration queue.

The following table lists the Message Queuing properties and the MQSeries fields to which they are converted.

Message Queuing properties	**Converted to MQSeries fields**
PROPID_M_DEST_QUEUE and PROPID_M_DEST_QUEUE_LEN	MQOD.ObjectName and MQOD.ObjectQMgrName
PROPID_M_RESP_QUEUE and PROPID_M_RESP_QUEUE_LEN	MQMD.ReplyToQ and MQMD.ReplyToQMgr
PROPID_M_ADMIN_QUEUE and PROPID_M_ADMIN_QUEUE_LEN	MQMD.ReplyToQ and MQMD.ReplyToQMgr

Message Class (PROPID_M_CLASS)

MSMQ-MQSeries Bridge converts the Message Queuing (also known as MSMQ) message class (PROPID_M_CLASS) to the MQSeries message type and feedback. It translates the message class values according to the following table.

Value of Message Queuing property	**Converted to MQSeries value**	
PROPID_M_CLASS	MQMD.MsgType	MQMD.Feedback
MQMSG_CLASS_NORMAL	MQMT_REQUEST or MQMT_DATAGRAM	MQFB_NONE
MQMSG_CLASS_ACK_REACH_QUEUE	MQMT_REPORT	MQFB_COA
MQMSG_CLASS_ACK_RECEIVE		MQFB_COD
MQMSG_CLASS_NACK_RECEIVE_TIMEOUT		MQFB_EXPIRATION
MQMSG_CLASS_NACK_REACH_QUEUE_TIMEOUT		MQFB_EXPIRATION
MQMSG_CLASS_NACK_Q_EXCEED_QUOTA		MQRC_Q_FULL
MQMSG_CLASS_NACK_ACCESS_DENIED		MQRC_NOT_AUTHORIZED
MQMSG_CLASS_NACK_ERROR		MQFB_APPL_TYPE_ERROR
Any other value		MQFB_NONE

MQMSG_CLASS_NORMAL is converted to **MQMD_REQUEST** if the message includes a response queue (**PROPID_M_RESP_QUEUE**) or if **PROPID_M_RESP_QUEUE** is missing, NULL, or an empty string.

Message Expiration (PROPID_M_TIME...)

Message Queuing (also known as MSMQ) provides two message expiration properties, **PROPID_M_TIME_TO_REACH_QUEUE** and **PROPID_M_TIME_TO_BE_RECEIVED**, both in units of seconds. MQSeries provides a single expiration field, **MQMD.Expiry**, whose units are tenths of a second.

MSMQ-MQSeries Bridge converts the values as listed in the following table.

Value of Message Queuing properties PROPID_M_TIME_TO_REACH_QUEUE and PROPID_M_TIME_TO_BE_RECEIVED	**Converted to MQSeries value of MQMD.Expiry**
Both values are INFINITE	MQEI_UNLIMITED
One or both values are not INFINITE	Ten times the smaller of the two Message Queuing values

Note

Message Queuing typically interprets **INFINITE** as 90 days. In practice, the MSMQ-MQSeries Bridge does not apply the **INFINITE** conversion because Message Queuing decrements the values slightly during transmission. If you need an **MQMD.Expiry** value of exactly **MQEI_UNLIMITED**, you should send this value in the message extension.

Message Acknowledgment (PROPID_M_ACKNOWLEDGE)

MSMQ-MQSeries Bridge supports the Message Queuing (also known as MSMQ) and MQSeries acknowledgment mechanisms. You can send a Message Queuing message to MQSeries and receive an automatic acknowledgment from the MQSeries Queue Manager.

To do this, set the Message Queuing acknowledgment property (**PROPID_M_ACKNOWLEDGE**) to a value that requests an acknowledgment. Also specify the administration or response queue name (**PROPID_M_ADMIN_QUEUE** or **PROPID_M_RESP_QUEUE**), to which the acknowledgment is sent. For more information about queue format names, see Queue Format Names (PROPID_M_..._QUEUE).

MSMQ-MQSeries Bridge converts the acknowledgment property to the MQSeries **MQMD.Report** field, as listed in the following table. When MQSeries receives the message, it returns the appropriate acknowledgment through MSMQ-MQSeries Bridge.

Value of Message Queuing property PROPID_M_ACKNOWLEDGE	**Converted to MQSeries value of MQMD.Report**
MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE	MQRO_EXCEPTION | MQRO_COA
MQMSG_ACKNOWLEDGMENT_FULL_RECEIVE	MQRO_EXCEPTION | MQRO_EXPIRATION | MQRO_COD
MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE	MQRO_EXCEPTION
MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE	MQRO_EXCEPTION | MQRO_EXPIRATION
MQMSG_ACKNOWLEDGMENT_NONE	MQRO_NONE

If both PROPID_M_ACKNOWLEDGE and PROPID_M_JOURNAL are included in a Message Queuing message, the value of the MQSeries **MQMD.Report** field is computed by a bitwise or of the values converted from the two Message Queuing properties. For more information about other Message Queuing properties, see Other Message Queuing Properties with Equivalent Properties.

If the **MSMQ_PROPID_M_ACKNOWLEDGE** property has a value of **MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE**, the value of MQSeries **MQMD.Report** field is computed by a bitwise or **of MQRO_EXCEPTION** and **MQRO_COA**.

If the **MSMQ_PROPID_M_ACKNOWLEDGE** property has a value of **MQMSG_ACKNOWLEDGMENT_FULL_RECEIVE**, the value of MQSeries MQMD.Report field is computed by a bitwise or of **MQRO_EXCEPTION**, **MQRO_EXPIRATION**, and **MQRO_COD**.

If the Message Queuing **PROPID_M_ACKNOWLEDGE** property has a value of **MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE**, the value of MQSeries MQMD.Report field is computed by a bitwise or of **MQRO_EXCEPTION** and **MQRO_EXPIRATION**.

Other Message Queuing Properties with Equivalent Properties

The following Message Queuing (also known as MSMQ) properties have MQSeries equivalents. The MSMQ-MQSeries Bridge converts the values of each property as listed in the table.

Note

To save space in the table, the prefixes PROPID_M_ and MQMD are omitted from the Message Queuing property names and the MQSeries field names, respectively. For example, the first row of data means that **PROPID_M_BODY_TYPE** is converted to MQMD.Format.

Message Queuing property		**Converted to MQSeries**	
PROPID_M_	**Value**	MQMD	**Value**
BODY_TYPE	VT_BSTR Any other value	Format	MQFMT_STRING MQFMT_NONE
CORRELATIONID	Value (20 bytes)	CorrelId	"FQ2Q" + value
DELIVERY	MQMSG_DELIVERY_ RECOVERABLE EXPRESS	Persistence	MQPER_ PERSISTENT NOT_PERSISTENT
JOURNAL	MQMSG_ DEADLETTER MQMSG_JOURNAL MQMSG_JOURNAL_NONE	Reporta	MQRO_ DEAD_LETTER_Q (Ignored) DISCARD_MSG
LABEL LABEL_LEN	Value	ApplIdentityData	Value (MQCHAR32)
MSGID	Value (20 bytes)	MsgId	"FQ2Q" + value
PRIORITY	0 1 2 3 4 5 6 7	Priority	1 3 4 5 6 7 8 9
SENTTIME	Seconds since Jan. 1, 1970	PutDate PutTime	YYYMMDD format HHMMSSTH format

If both **PROPID_M_ACKNOWLEDGE** and **PROPID_M_JOURNAL** are included in a Message Queuing message, the **MQSeries MQMD.Report** field is computed by a bitwise or of the values converted from the two Message Queuing queuing properties.

Unconverted Properties

The following Message Queuing (also known as MSMQ) properties have no equivalents in MQSeries. The MSMQ-MQSeries Bridge ignores these properties and does not transmit them to MQSeries:

- **PROPID_M_APPSPECIFIC**
- **PROPID_M_ARRIVEDTIME**
- **PROPID_M_AUTH_LEVEL**
- **PROPID_M_AUTHENTICATED**
- **PROPID_M_CONNECTOR_TYPE**
- **PROPID_M_DEST_SYMM_KEY**
- **PROPID_M_DEST_SYMM_KEY_LEN**
- **PROPID_M_ENCRYPTION_ALG**
- **PROPID_M_HASH_ALG**
- **PROPID_M_PRIV_LEVEL**
- **PROPID_M_PROV_NAME**
- **PROPID_M_PROV_NAME_LEN**
- **PROPID_M_PROV_TYPE**
- **PROPID_M_SECURITY_CONTEXT**
- **PROPID_M_SENDER_CERT**
- **PROPID_M_SENDER_CERT_LEN**
- **PROPID_M_SENDERID**
- **PROPID_M_SENDERID_LEN**
- **PROPID_M_SENDERID_TYPE**
- **PROPID_M_SIGNATURE**
- **PROPID_M_SIGNATURE_LEN**
- **PROPID_M_SRC_MACHINE_ID**
- **PROPID_M_TRACE**
- **PROPID_M_VERSION**

Transaction Properties

The following Message Queuing (also known as MSMQ) properties are not converted to MQSeries fields or values:

- **PROPID_M_XACT_STATUS_QUEUE**
- **PROPID_M_XACT_STATUS_QUEUE_LEN**

For more information about transactions, see Transaction Support Using MSMQ-MQSeries Bridge.

Building an MQSeries Message

This section explains how the MSMQ-MQSeries Bridge builds a complete MQSeries message, including all needed fields whether or not they have Message Queuing (also known as MSMQ) equivalents. The conversion to MQSeries fields from Message Queuing properties is listed from the MQSeries perspective.

For information about these same conversion rules from the Message Queuing perspective, see Converting Message Queuing Properties.

In This Section

- Message Buffer
- Object Descriptor (MQOD)
- Message Descriptor (MQMD)

Message Buffer

The MQPUT or MQGET message buffer of MQSeries is equivalent to the message body property of Message Queuing (also known as MSMQ). The length of the MQPUT buffer is the Message Queuing message body size.

The following table lists the MQSeries fields and the Message Queuing properties from which they are converted.

MQSeries fields	Converted from Message Queuing property
Message buffer	PROPID_M_BODY
Message buffer length	PROPID_M_BODY_SIZE

Object Descriptor (MQOD)

The MSMQ-MQSeries Bridge retrieves the MQSeries object descriptor fields from the Message Queuing (also known as MSMQ) format name of the destination queue.

The following table lists the MQSeries fields and the Message Queuing properties from which they are converted.

MQSeries fields	Converted from Message Queuing property
MQOD.ObjectName and MQOD.ObjectQMgrName	PROPID_M_DEST_QUEUE and PROPID_M_DEST_QUEUE_LEN

Message Descriptor (MQMD)

Some fields of the MQSeries message descriptor have no equivalent in Message Queuing (also known as MSMQ). The MSMQ-MQSeries Bridge assigns default values to these fields. Alternatively, you can pass explicit values of the MQMD fields using the Message Queuing message extension property (**PROPID_M_EXTENSION**).

The following table lists the MQSeries fields and the default value assigned by MSMQ-MQSeries Bridge.

MQSeries MQMD fields	Default value assigned by MSMQ-MQSeries Bridge
MQMD.AccountingToken	MQACT_NONE
MQMD.CodeCharSetId	MQCCSI_Q_MGR
MQMD.Encoding	MQENC_NATIVE
MQMD.PutApplName	NULL
MQMD.PutApplType	NULL
MQMD.StrucId	MQMD_STRUC_ID
MQMD.Version	MQMD_VERSION_1

Some fields of the MQSeries message descriptor have no equivalent in Message Queuing and the MSMQ-MQSeries Bridge does not assign default values to these fields. You can assign explicit values for these MQMD fields using the Message Queuing message extension property (**PROPID_M_EXTENSION**).

The following table lists the MQSeries fields for which no default value is assigned by MSMQ-MQSeries Bridge.

MQSeries MQMD fields	Default value assigned by MSMQ-MQSeries Bridge
MQMD.UserIdentifier	No default value is assigned

Many of the MQMD fields are equivalent to one or more Message Queuing properties. The conversion rules are listed in the following table.

You can override the conversion rules by passing explicit MQMD values in the Message Queuing message extension property.

To save space in the table, the prefix MQMD. is omitted from the MQSeries field names. For example, the first row of data means that MQMD.ApplIdentityData is built from the Message Queuing properties **PROPID_M_LABEL** and **PROPID_M_LABEL_LEN**.

MQSeries MQMD field	MQSeries MQMD field value	Converted from Message Queuing property
ApplIdentity	Value (MQCHAR32)	The value of PROPID_M_LABEL and PROPID_M_LABEL_LEN
CorrelId	"FQ2Q" + value	The value of PROPID_M_CORRELATIONID (20 bytes)
Expiry	MQEI_UNLIMITED	Both PROPID_M_TIME_TO_BE_RECEIVED and PROPID_M_TIME_TO_REACH_QUEUE have value of INFINITE
Expiry	10 times the smaller of the two Message Queuing values	PROPID_M_TIME_TO_BE_RECEIVED or PROPID_M_TIME_TO_REACH_QUEUE have a value that is not INFINITE
Feedback	MQFB_EXPIRATION	PROPID_M_CLASS has a value of MQMSG_CLASS_NACK_RECEIVE_TIMEOUT
Feedback	MQFB_APPL_TYPE_ERROR	PROPID_M_CLASS has a value of MQMSG_CLASS_NACK_REACH_QUEUE_TIMEOUT
Feedback	MQRC_Q_FULL	PROPID_M_CLASS has a value of MQMSG_CLASS_NACK_Q_EXCEED_QUOTA
Feedback	MQRC_NOT_AUTHORIZED	PROPID_M_CLASS has a value of MQMSG_CLASS_NACK_ACCESS_DENIED
Feedback	MQFB_COA	PROPID_M_CLASS has a value of MQMSG_CLASS_ACK_REACH_QUEUE

Feedback	MQFB_COD	PROPID_M_CLASS has a value of MQMSG_CLASS_ACK_RECEIVE
Feedback	MQFB_NONE	PROPID_M_CLASS has any other value
Format	MQFMT_STRING	PROPID_M_BODY_TYPE has a value of VT_BSTR
Format	MQFMT_NONE	PROPID_M_BODY_TYPE has any other value
MsgId	"FQ2Q" + value	The value of PROPID_M_MSGID (20 bytes)
MsgType	MQMT_DATAGRAM	PROPID_M_CLASS has a value of MQMSG_CLASS_NORMAL
MsgType	MQMT_REQUESTREPORT	PROPID_M_CLASS has any other value
Persistence	MQPER_PERSISTENT	PROPID_M_DELIVERY has a value of MQMSG_DELIVERY_RECOVERABLE
Persistence	MQPER_NOT_PERSISTENT	PROPID_M_DELIVERY has a value of MQMSG_DELIVERY_EXPRESS
Priority	1	PROPID_M_PRIORITY has a value of 0
Priority	3	PROPID_M_PRIORITY has a value of 1
Priority	4	PROPID_M_PRIORITY has a value of 2
Priority	5	PROPID_M_PRIORITY has a value of 3
Priority	6	PROPID_M_PRIORITY has a value of 4
Priority	7	PROPID_M_PRIORITY has a value of 5
Priority	8	PROPID_M_PRIORITY has a value of 6
Priority	9	PROPID_M_PRIORITY has a value of 7
PutDate	YYYMMDD format	The date from the Message Queuing PROPID_M_SENTTIME property which has a value of seconds since January 1, 1970
PutTime	HHMMSSTH format	The time from the Message Queuing PROPID_M_SENTTIME property which has a value of seconds since January 1, 1970
ReplyToQ	Retrieved from Message Queuing format name	The Message Queuing format name from PROPID_M_RESP_QUEUE if this property is included and is not NULL or an empty string
ReplyToQ	Retrieved from Message Queuing format name	The Message Queuing format name from PROPID_M_ADMIN_QUEUE if PROPID_M_RESP_QUEUE is not included or is NULL or an empty string
ReplyToQMgr	Retrieved from Message Queuing format name	The Message Queuing format name from PROPID_M_RESP_QUEUE if this property is included and is not NULL or an empty string
ReplyToQMgr	Retrieved from Message Queuing format name	The Message Queuing format name from PROPID_M_ADMIN_QUEUE if PROPID_M_RESP_QUEUE is not included or is NULL or an empty string
Report	MQRO_EXCEPTION|COA	PROPID_M_ACKNOWLEDGE has a value of MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE
Report	MQRO_EXCEPTION|EXPIRATION|COD	PROPID_M_ACKNOWLEDGE has a value of MQMSG_ACKNOWLEDGMENT_FULL_RECEIVE
Report	MQRO_EXCEPTION	PROPID_M_ACKNOWLEDGE has a value of MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE
Report	MQRO_EXCEPTION|EXPIRATION	PROPID_M_ACKNOWLEDGE has a value of MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE

Report	MQRO_NONE	PROPID_M_ACKNOWLEDGE has a value of MQMSG_ACKNOWLEDGMENT_NONE
Report	MQRO_DEAD_LETTER_Q	PROPID_M_JOURNAL has a value of MQMSG_DEADLETTER
Report	The Message Queuing value is ignored	PROPID_M_JOURNAL has a value of MQMSG_JOURNAL
Report	MQRO_DISCARD_MSG	PROPID_M_JOURNAL has a value of MQMSG_JOURNAL_NONE

In practice, the MSMQ-MQSeries Bridge does not assign **MQEI_UNLIMITED** as value for MQMD.Expiry because Message Queuing interprets INFINITE values typically as 90 days and decrements them slightly during transmission. To assign an MQMD.Expiry value of exactly **MQEI_UNLIMITED**, send this value in the message extension.

The MSMQ-MQSeries Bridge assigns **MQMT_DATAGRAM** as value for MQMD.MsgType if the Message Queuing **PROPID_M_RESP_QUEUE** is missing, NULL, or an empty string. Otherwise a value of **MQMT_REQUEST** is assigned to MQMD.MsgType.

If both **PROPID_M_ACKNOWLEDGE** and **PROPID_M_JOURNAL** are included in a Message Queing message, MQMD.Report is computed by a bitwise or of the values converted from the two Message Queuing properties.

Converting Messages Sent from MQSeries to Message Queuing

When you send a message from IBM MQSeries to Microsoft® Message Queuing (also known as MSMQ), the MSMQ-MQSeries Bridge converts the message from an MQSeries data structure to a Message Queuing message property. o do this, MSMQ-MQSeries Bridge maps the various data fields of the MQSeries message as nearly as possible to equivalent Message Queuing message properties.

This section describes the conversion rules by which this is done. The information in this section applies to messages that you send from MQSeries to Message Queuing. For messages sent from Message Queuing to MQSeries, see Converting Messages Sent from Message Queuing to MQSeries.

The section contains two main subsections, which provide essentially the same information, but from complementary points of view. The first section describes the conversion rules from the sender's point of view, and explains how the MSMQ-MQSeries Bridge converts each MQSeries field that you include in a message. The second section explains the rules from the receiver's point of view. Use this section to learn how MSMQ-MQSeries Bridge builds a complete Message Queuing message containing all the needed properties, whether or not they have exact MQSeries equivalents.

Besides the conversions described in this section, the MSMQ-MQSeries Bridge transmits the original MQSeries message descriptor fields in the Message Queuing message extension property (**PROPID_M_EXTENSION**).

In This Section

- Converting MQSeries Fields
- Building a Message Queuing Message

Converting MQSeries Fields

This section explains how the MSMQ-MQSeries Bridge converts the fields of an MQSeries message to Message Queuing (also known as MSMQ). For information about Message Queuing properties that have no MQSeries equivalents, see Building a Message Queuing Message.

In This Section

Message Buffer

The MQGET or MQPUT buffer of MQSeries is equivalent to the message body property of Message Queuing (also known as MSMQ).

The following table lists the MQSeries fields and the Message Queuing properties to which they are converted.

MQSeries fields	Converted to Message Queuing property
Message buffer	PROPID_M_BODY
Message buffer length	PROPID_M_BODY_SIZE

Object Descriptor (MQOD)

The MSMQ-MQSeries Bridge converts the MQSeries remote Queue Manager and queue names to the Message Queuing (also known as MSMQ) destination queue name. The MQSeries fields are interpreted as a Message Queuing format name or path name.

The following table lists the MQSeries fields and the Message Queuing properties that they are converted to.

MQSeries fields	**Converted to Message Queuing property**
MQOD.ObjectName and MQOD.ObjectQMgrName	PROPID_M_DEST_QUEUE and PROPID_M_DEST_QUEUE_LEN

Character Substitutions in Object Descriptor Conversion

Certain characters are supported in Message Queuing (also known as MSMQ) format names, but not in MQSeries names. When you assign the MQSeries MQOD.ObjectName, the MSMQ-MQSeries Bridge performs character substitutions to the Message Queuing format names. The following table lists special characters in MQSeries names and what MSMQ-MQSeries Bridge converts these characters to in Message Queuing format names.

Characters in MQSeries MQOD field names	**Characters substituted in Message Queuing format names**
/ (forward slash))	: (colon) or \ (backslash)
_ (underscore)	- (hyphen) or = (equal)
P_ (at start of MQOD.ObjectName)	PRIVATE=

The MSMQ-MQSeries Bridge converts the characters back when it transmits the MQSeries message to Message Queuing. The characters are interpreted according to context to generate a legal Message Queuing name. For example, MSMQ-MQSeries Bridge converts the format name:

```
DIRECT_OS/MACHINE2/QUEUE4
```

to the following:

```
DIRECT=OS:MACHINE2\QUEUE4.
```

Note that this character substitution is only for format names, not for computer or path names. If you want to address Message Queuing path names, do not include hyphens or other characters not supported by MQSeries. For example, the path name:

```
MACHINE2\MY-QUEUE
```

is legal in Message Queuing, but you cannot specify the hyphen character in MQSeries.

Format Name Method of Object Descriptor Conversion

Subject to the following conditions, the MSMQ-MQSeries Bridge interprets the MQSeries MQOD.ObjectName field as the format name of the Message Queuing (also known as MSMQ) destination queue:

- The MQOD.ObjectQMgrName is an MQSeries alias for a Message Queuing computer.
- The MQOD.ObjectName begins with PUBLIC_, P_, or DIRECT_OS/.

If these conditions apply, the MSMQ-MQSeries Bridge converts the MQSeries MQOD.ObjectName to the Message Queuing format name of the destination queue using one of the following methods based on the value of **MQOD.ObjectName**:

```
PUBLIC=<GUID>
PRIVATE=<machine GUID>\<file number>
DIRECT=OS:<path name>.
```

If you know the globally unique identifier (GUID) of the destination queue, the PUBLIC=<GUID> syntax gives better performance than the DIRECT=OS:<path name> syntax. Also note that nontransacted private queues are supported only on the computer running the MSMQ-MQSeries Bridge.

Path Name Method of Object Descriptor Conversion

If the conditions for the format name method do not hold, the MSMQ-MQSeries Bridge interprets MQOD.ObjectQMgrName\MQOD.ObjectName as a Message Queuing (also known as MSMQ) path name. The remote Queue Manager name must be an alias for a computer running Message Queuing that you have previously defined in MQSeries.

For example, if MQOD.ObjectQMgrName is MACHINE2 and MQOD.ObjectName is QUEUE4, the MSMQ-MQSeries Bridge sends the message to a Message Queuing queue having the path name MACHINE2\QUEUE4.

The MSMQ-MQSeries Bridge determines the Message Queuing format name corresponding to this path name to forward the message.

Note
The MQSeries name fields are limited to 48 characters each. If the path name is longer than this, use the format name method instead.

Queue Alias Method of Object Descriptor Conversion

Optionally, you can address a Message Queuing (also known as MSMQ) queue using an MQSeries queue alias. If you use this method, set MQOD.ObjectName to the queue alias. Leave MQOD.ObjectQMgrName blank, or set it to the MQSeries Queue Manager where the transmission queue is located.

Examples of Object Descriptor Conversion

In the following examples, you want to send a message to a Message Queuing (also known as MSMQ) destination having the following identifiers:

```
Machine name = MACHINE2
Queue name = QUEUE4
GUID = A56F41B4-9869-11D0-AF8F-0000E8D1C3A7
```

MSMQ-MQSeries Bridge is installed on a computer called BRIDGEMQ1. You have defined the aliases BRIDGEMQ1 and BRIDGEMQ1% for this computer in MQSeries. In the MSMQ-MQSeries Bridge Manager, you have configured BRIDGEMQ1 for MQS->Message Queuing message pipe and BRIDGEMQ1% for MQS->Message Queuing transactional message pipe.

You may address a message to this queue in any of the following ways:

Format name method

Using the format name method and MQS->Message Queuing message pipe, the MQSeries name would become:

```
MQOD.ObjectQMgrName = "BRIDGEMQ1"
MQOD.ObjectName = "PUBLIC_A56F41B4_9869_11D0_AF8F_0000E8D1C3A7"
```

- or -

```
MQOD.ObjectQMgrName = "BRIDGEMQ1"
MQOD.ObjectName = "DIRECT_OS/MACHINE2/QUEUE4"
```

Using the format name method and MQS->Message Queuing transactional message pipe, the MQSeries name would become:

```
MQOD.ObjectQMgrName = "BRIDGEMQ1%"
MQOD.ObjectName = "PUBLIC_A56F41B4_9869_11D0_AF8F_0000E8D1C3A7"
```

- or -

```
MQOD.ObjectQMgrName = "BRIDGEMQ1%"
MQOD.ObjectName = "DIRECT_OS/MACHINE2/QUEUE4"
```

Optionally, you can define the MQSeries aliases MACHINE2 and MACHINE2% for the Message Queuing destination computer. You now have two additional ways to address the queue.

Using the path name method and MQS->Message Queuing message pipe, the MQSeries name would become:

```
MQOD.ObjectQMgrName = "MACHINE2"
MQOD.ObjectName = "QUEUE4"
```

Using the path name method and MQS->Message Queuing transactional message pipe, the MQSeries name would become:

```
MQOD.ObjectQMgrName = "MACHINE2%";
MQOD.ObjectName = "QUEUE4"
```

In yet another option, you can define the MQSeries aliases QUEUE4 and QUEUE4% for the Message Queuing destination queues. If you do this, you can address the queue using the following syntax.

Using the queue alias method and normal service, the MQSeries name would become:

```
MQOD.ObjectQMgrName = ""
MQOD.ObjectName = "QUEUE4"
```

Using the queue alias method and high service, the MQSeries name would become:

```
MQOD.ObjectQMgrName = ""
MQOD.ObjectName = "QUEUE4%"
```

Message Descriptor (MQMD)

The MSMQ-MQSeries Bridge converts most of the MQSeries field values to Message Queuing (also known as MSMQ) property values.

MQMD.Report Field

The MSMQ-MQSeries Bridge supports the MQSeries and Message Queuing (also known as MSMQ) acknowledgment mechanisms. You can send a Message Queuing message to MQSeries and receive an automatic acknowledgment from the Message Queuing Queue Manager.

To do this, set the MQMD.Report field of the MQSeries message to a value that requests an acknowledgment. Also set the MQMD.ReplyToQ and MQMD.ReplyToQMgr fields, which specify where the acknowledgment is sent.

The MSMQ-MQSeries Bridge converts MQMD.Report to the Message Queuing acknowledgment property. When Message Queuing receives the message, it returns the appropriate acknowledgment thorough the MSMQ-MQSeries Bridge.

The MSMQ-MQSeries Bridge also supports the MQSeries and Message Queuing dead letter mechanism. For this purpose, MSMQ-MQSeries Bridge converts the MQMD.Report values to the Message Queuing journaling property.

The conversions are listed in the following table.

Value of MQSeries field MQMD.Report	**Converted to Message Queuing property and value**
MQRO_NONE	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NONE
MQRO_EXCEPTION	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE
MQRO_EXCEPTION_WITH_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE
MQRO_EXCEPTION_WITH_FULL_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_REACH_QUEUE
MQRO_EXPIRATION	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE
MQRO_EXPIRATION_WITH_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE
MQRO_EXPIRATION_WITH_FULL_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_NACK_RECEIVE
MQRO_COA	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE
MQRO_COA_WITH_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE
MQRO_COA_WITH_FULL_DATA (Note 1)	The value of PROPID_M_ACKNOWLEDGE is set to MQMSG_ACKNOWLEDGMENT_FULL_REACH_QUEUE
MQRO_DEAD_LETTER_Q	The value of PROPID_M_JOURNAL is set to MQMSG_DEADLETTER
MQRO_DISCARD_MSG	The value of PROPID_M_JOURNAL is set to MQMSG_JOURNAL_NONE
MQRO_NEW_MSG_ID	(Not converted)
MQRO_PASS_MSG_ID	(Not converted)
MQRO_COPY_MSG_ID_TO_CORREL_ID	(Not converted)
MQRO_PASS_CORREL_ID	(Not converted)

Note

For these report values, the MSMQ-MQSeries Bridge duplicates part or all of the message buffer in the Message Queuing message extension (**PROPID_M_EXTENSION**). The increased message size may degrade performance.

MQMD.MsgType and MQMD.Feedback Fields

The values of the MQSeries MQMD.MsgType and MQMD.Feedback fields are converted to the Message Queuing (also known as MSMQ) message class property.

The conversions are listed in the following table based on the value of MQMD.MsgType when MQMD.Feedback is MQFB_NONE.

Value of MQSeries MQMD.MsgType	Converted to value of Message Queuing PROP_M_CLASS
MQMT_SYSTEM_FIRST	MQMSG_CLASS_NORMAL
MQMT_SYSTEM_LAST	MQMSG_CLASS_NORMAL
MQMT_DATAGRAM	MQMSG_CLASS_NORMAL
MQMT_REQUEST	MQMSG_CLASS_NORMAL
MQMT_REPLY	MQMSG_CLASS_NORMAL
MQMT_APPL_FIRST	MQMSG_CLASS_NORMAL
MQMT_APPL_LAST	MQMSG_CLASS_NORMAL

The conversions are listed in the following table based on the value of MQMD.Feedback when MQMD.MsgType is MQMT_REPORT.

Value of MQSeries MQMD.Feedback	Converted to value of Message Queuing PROP_M_CLASS
MQFB_EXPIRATION	MQMSG_CLASS_NACK_RECEIVE_TIMEOUT
MQFB_COA	MQMSG_CLASS_ACK_REACH_QUEUE
MQFB_COD	MQMSG_CLASS_ACK_RECEIVE
MQFB_APPL_TYPE_ERROR	MQMSG_CLASS_NACK_ERROR
MQFB_DATA_LENGTH_ZERO	MQMSG_CLASS_NACK_ERROR
MQFB_DATA_LENGTH_NEGATIVE	MQMSG_CLASS_NACK_ERROR
MQFB_DATA_LENGTH_TOO_BIG	MQMSG_CLASS_NACK_ERROR
MQFB_BUFFER_OVERFLOW	MQMSG_CLASS_NACK_ERROR
MQFB_LENGTH_OFF_BY_ONE	MQMSG_CLASS_NACK_ERROR
MQFB_NONE	(Not converted)
MQFB_SYSTEM_FIRST	(Not converted)
MQFB_SYSTEM_LAST	(Not converted)
MQFB_APPL_FIRST	(Not converted)
MQFB_APPL_LAST	(Not converted)
MQFB_TM_ERROR	(Not converted)
MQFB_IIH_ERROR	(Not converted)
MQFB_NOT_AUTHORIZED_FOR_IMS	(Not converted)
MQFB_IMS_ERROR	(Not converted)
MQFB_IMS_FIRST	(Not converted)

MQFB_IMS_LAST	(Not converted)
MQFB_QUIT	(Not supported)
MQRC_NOT_AUTHORIZED	MQMSG_CLASS_NACK_ACCESS_DENIED
MQRC_Q_FULL	MQMSG_CLASS_NACK_Q_EXCEED_QUOTA
MQRC_PERSISTENT_NOT_ALLOWED	MQMSG_CLASS_NACK_ERROR
MQRC_MSG_TOO_BIG_FOR_Q_MGR	MQMSG_CLASS_NACK_ERROR
MQRC_MSG_TOO_BIG_FOR_Q	MQMSG_CLASS_NACK_ERROR
MQRC_PUT_INHBITED	(Not converted)

MQMD.ReplyToQ and MQMD.ReplyToQMgr Fields

The MSMQ-MQSeries Bridge converts the MQSeries ReplyToQMgr and ReplyToQ fields to a Message Queuing (also known as MSMQ) format name. The name is assigned both to the response queue property and to the administration queue property of the new Message Queuing message.

The MSMQ-MQSeries Bridge interprets the **ReplyToQMgr** and the **ReplytoQ** fields in the same way as the destination queue name.

If MQMD.ReplyToQMgr is the MSMQ-MQSeries Bridge computer name and MQMD.ReplyToQ begins with PUBLIC_, P_, or DIRECT_OS/, MSMQ-MQSeries Bridge interprets MQMD.ReplyToQ as a Message Queuing format name.

Otherwise, the MSMQ-MQSeries Bridge interprets MQMD.ReplyToQMgr\MQMD.ReplyToQ as a Message Queuing path name and determines the Message Queuing format name.

The following table lists the MQSeries fields and the Message Queuing properties to which they are converted.

MQSeries fields	**Converted to Message Queuing property**
MQOD.ReplyToQMgr and MQOD.ReplyToQ	PROPID_M_RESP_QUEUE
MQOD.ReplyToQMgr and MQOD.ReplyToQ	PROPID_M_ADMIN_QUEUE (same value as PROPID_M_RESP_QUEUE)

Other MQMD Fields

The following table lists the conversions of additional MQMD fields, in addition to the ones described previously, to Message Queuing properties.

Note
To save space in the table, the prefixes MQMD and PROPID_M_ are omitted from the MQSeries field names and the Message Queuing property names, respectively. For example, the first row of data means that MQMD.ApplIdentityData is converted to **PROPID_M_LABEL** and **PROPID_M_LABEL_LEN**.

MQSeries field		**Converted to Messa ge Queuing**	
MQMD.	**Value**	**PROPID_M_**	**Value**
ApplIdenti tyData	Value (MQCHAR32)	LABEL	Value
CorrelId	Value (MQBYTE24)	CORRELATIONID	Last 20 bytes of value
Expiry	MQEI_UNLIMITED Value > 0 (t enths of seconds)	TIME_TO_BE_RECEIVE D	INFINITE Value/10 (seconds)
Format	MQFMT_STRING Any other va lue	BODY_TYPE	VT_BSTR Not converted
Persistenc e	MQPER_ PERSISTENT NOT_PE RSISTENT	DELIVERY	MQMSG_DELIVERY_ RECOVERABLE EXPRESS
Priority	0, 1 2, 3 4 5 6 7 8 9	PRIORITY	0 1 2 3 4 5 6 7
UserIdenti fier	Value	SENDERID	SID value (if the user is registered in Windows 2000 Not conv erted (if the user is not registered)

Unconverted MQSeries MQMD Fields

The following MQSeries fields have no equivalents in Message Queuing (also known as MSMQ). The MSMQ-MQSeries Bridge ignores these fields and does not transmit them to Message Queuing.

Like all MQMD fields, MSMQ-MQSeries Bridge stores the fields in the Message Queuing message extension property:

- MQMD.AccountingToken
- MQMD.ApplOriginData
- MQMD.BackoutCount
- MQMD.Encoding
- MQMD.MsgId
- MQMD.PutApplName
- MQMD.PutApplType
- MQMD.PutDate
- MQMD.PutTime
- MQMD.StrucId

When the value of MQMD.CodeCharSetID is **MQCCSI_Q_QMG**, the MSMQ-MQSeries Bridge ignores this field and does not transmit it to Message Queuing. When the value of MQMD.CodeCharSetID is **MQCCSI_Q_QMG**, The MSMQ-MQSeries Bridge does not support this value.

When the value of MQMD.Version is **MQPMO_VERSION_1**, the MSMQ-MQSeries Bridge ignores this field and does not transmit it to Message Queuing.

Building a Message Queuing Message

This section explains how the MSMQ-MQSeries Bridge builds a complete Message Queuing (also known as MSMQ) message, including all needed properties, whether or not they have MQSeries equivalents.

Message Body (PROPID_M_BODY)

The Message Queuing (also known as MSMQ) message body is equivalent to the MQPUT or MQGET message buffer of MQSeries. The length of the MQPUT buffer is the Message Queuing message body size.

The following table lists the Message Queuing properties and the MQSeries fields from which they are converted.

Message Queuing property	Converted from MQSeries field
PROPID_M_BODY	Message buffer
PROPID_M_BODY_SIZE	Message buffer length

Queue Format Names (PROPID_M_..._QUEUE)

MSMQ-MQSeries Bridge retrieves the Message Queuing (also known as MSMQ) format names of the destination, response, and administration queues from the MQSeries queue and Queue Manager names.

The following table lists the Message Queuing properties and the MQSeries fields from which they are converted.

Message Queuing properties	Converted from MQSeries fields
PROPID_M_DEST_QUEUE and PROPID_M_DEST_QUEUE_LEN	MQOD.ObjectName and MQOD.ObjectQMgrName
PROPID_M_RESP_QUEUE and PROPID_M_RESP_QUEUE_LEN	MQMD.ReplyToQ and MQMD.ReplyToQMgr
PROPID_M_ADMIN_QUEUE and PROPID_M_ADMIN_QUEUE_LEN	MQMD.ReplyToQ and MQMD.ReplyToQMgr

Message Class (PROPID_M_CLASS)

The MSMQ-MQSeries Bridge assigns the Message Queuing (also known as MSMQ) message class based on the MQSeries MQMD.MsgType and MQMD.Feedback values. For detailed information, see MQMD.MsgType and MQMD.Feedback Fields.

The following table lists the Message Queuing properties and the MQSeries fields from which they are converted.

Message Queuing properties	Converted from MQSeries fields
PROPID_M_CLASS	MQMD.MsgType and MQMD.Feedback

Message Expiration (PROPID_M_TIME...)

The MSMQ-MQSeries Bridge converts the MQSeries MQMD.Expiry value (in tenths of seconds) to the Message Queuing (also known as MSMQ) **PROPID_M_TIME_TO_BE_RECEIVED** (in seconds). If MQMD.Expiry is set to **MQEI_UNLIMITED**, the value of Message Queuing **PROPID_M_TIME_TO_BE_RECEIVED** is set to INFINITE. For other values of MQMD.Expiry greater than zero, the value of Message Queuing **PROPID_M_TIME_TO_BE_RECEIVED** is converted by dividing MQMD.Expiry by 10. The MSMQ-MQSeries Bridge does not set **PROPID_M_TIME_TO_REACH_QUEUE**.

Value of Message Queuing properties	Converted from MQSeries fields
PROPID_M_TIME_TO_BE_RECEIVED	MQMD.Expiry

Message Acknowledgment and Journaling (PROPID_M_ACKNOWLEDGE, PROPID_M_JOURNAL)

The Message Queuing (also known as MSMQ) message acknowledgment and journaling are converted from values of MQMD.Report.

Message Queuing property	Converted from MQSeries
PROPID_M_ACKNOWLEDGE PROPID_M_JOURNAL	MQMD.Report

Other Message Queuing Properties (PROPID_M_...)

Some Message Queuing (also known as MSMQ) properties have no equivalent in MQSeries. The MSMQ-MQSeries Bridge

assigns values to these properties as listed in the following table.

Message Queuing property	Value assigned by MSMQ-MQSeries Bridge
PROPID_M_APPSPECIFIC	None
PROPID_M_AUTH_LEVEL	Default
PROPID_M_CONNECTOR_TYPE	None
PROPID_M_DEST_QUEUE	Default
PROPID_M_DEST_QUEUE_LEN	Default
PROPID_M_DEST_SYMM_KEY	None
PROPID_M_ENCRYPTION_ALG	None
PROPID_M_HASH_ALG	None
PROPID_M_MSGID	Assigned by Message Queuing
PROPID_M_PRIV_LEVEL	Default
PROPID_M_PROV_NAME	None
PROPID_M_PROV_TYPE	None
PROPID_M_SECURITY_CONTEXT	Default
PROPID_M_SENDER_CERT	Default
PROPID_M_SENTTIME	Time when MSMQ-MQSeries Bridge transmits the message to Message Queuing
PROPID_M_SIGNATURE	None
PROPID_M_SRC_MACHINE_ID	GUID of the MSMQ-MQSeries Bridge machine
PROPID_M_TRACE	Default
PROPID_M_VERSION	0x0010

Equivalent Message Queuing Properties

The following Message Queuing (also known as MSMQ) properties have MQSeries equivalents. The MSMQ-MQSeries Bridge converts the values of each property as listed in the table.

Note

To save space in the table, the prefixes PROPID_M_ and MQMD. are omitted from the Message Queuing property names and the MQSeries field names, respectively. For example, the first row of data means that PROPID_M_BODY_TYPE is converted fr om MQMD.Format.

Message Que uing property		Converted fr om MQSerie s	
PROPID_M_	**Value**	**MQMD.**	**Value**

BODY_TYPE	VT_BSTR Not converted (default)	Format	MQFMT_STRING Any other value
CORRELATION ID	Last 20 bytes of value	MsgId	Value (MQBYTE24)
DELIVERY	MQMSG_DELIVERY_ RECOVERABLE EXPRESS	Persistence	MQPER_ PERSISTENT NOT_PERSISTENT
LABEL LABEL_L EN	Value	ApplIdentityD ata	Value (MQCHAR32)
PRIORITY	0 1 2 3 4 5 6 7	Priority	0, 1 2, 3 4 5 6 7 8 9
SENDERID SE NDERID_TYPE	SID value MQMSG_SENDERID_TYPE_ SID Not converted MQMSG_SENDERID_TYPE_ NONE	UserIdentifier	Value, if the user is registered in Windows 2000 If the user is not registered in Windows 2000

MSMQ-MQSeries Bridge Extensions Mechanism

Besides the automatic conversion of Microsoft® Message Queuing (also known as MSMQ) and IBM MQSeries messages, the MSMQ-MQSeries Bridge provides a mechanism for sending and receiving explicit MQSeries field values. The MSMQ-MQSeries Bridge Extension Property API supports the message extension property (**PROPID_M_EXTENSION**) of the Message Queuing server. The message extension property provides a way for applications to attach any type of data—in essence, custom properties—to a Message Queuing message.

Similar to the Message Queuing message body property (**PROPID_M_BODY**), the message extension can have any length. However, the message extension has a defined structure that lets an application label its data with a globally unique identifier (GUID) code. Applications can attach multiple extension fields, each labeled with its own GUID and all included in a single message extension property. This is done using the Message Queuing message extension property (**PROPID_M_EXTENSION**).

In This Section

Data Structure of a Message Extension

A message extension is a Message Queuing (also known as MSMQ) message property of arbitrary length. The property symbol of a message extension is **PROPID_M_EXTENSION** and its type indicator is VT_UI1|VT_VECTOR. A message extension is a sequential buffer containing any number of Message Queuing extension fields.

Each extension field comprises three subfields, as listed in the following table.

Subfield	Length of subfield (bytes)	Description
GUID	16	A GUID identifier, typically of the application that created the extension field.
Length	4	The length of the data subfield in bytes.
Data	Value of the length subfield	Any data that is part of the message extension.

For use with the MSMQ-MQSeries Bridge, the GUID identifier is set to the MSMQ-MQSeries Bridge GUID value.

The Message Queuing message extension length property, **PROPID_M_EXTENSION_LEN**, is of type indicator VT_UI4 and represents the overall size in bytes of all message extensions attached to a message. Message Queuing sets the message extension length automatically when you send a message. When you receive or peek at a message, you can look into the message extension length to detect whether the message contains any message extension fields and to determine the necessary receive buffer size.

The MSMQ-MQSeries Bridge Extension API functions work with an alternative data representation for a message extension, called an **EP** object. An **EP** object contains the same fields and subfields as a message extension, but in a format adapted for programming.

How MSMQ-MQSeries Bridge Creates a Message Extension

When the MSMQ-MQSeries Bridge processes a message from MQSeries, it creates a **PROPID_M_EXTENSION** property, which it includes in the message that it transmits to Message Queuing (also known as MSMQ).

In This Section

MQMD Extension Field

Ordinarily, the message extension contains a single extension field with the following structure:

- The MSMQ-MQSeries Bridge globally unique identifier (GUID) code is stored in the GUID subfield of the extension.
- The size of MQMD is stored in the length subfield of the extension.
- The MQMD is copied byte-for-byte into the data buffer of the extension.

The MSMQ-MQSeries Bridge GUID is the value of the sg_MSMQExtMQMD constant which is defined in the Mqsrext.h include file found in the SDK\Include subdirectory.

The MQMD extension has the following GUID for MQMD version 2.

```
static const GUID sg_MSMQExtMQMDE =
{ 0x18ae68f5, 0x989b, 0x11d3,
    { 0x8d, 0xf9, 0x0, 0x0, 0xf8, 0x1a,  0xea, 0x1f }
};
```

Error Extension Field

If the MSMQ-MQSeries Bridge encounters an MQSeries error when it transmits a message from Message Queuing (also known as MSMQ), it records the error in the extension property and places the message on the dead letter queue.

To do this, the MSMQ-MQSeries Bridge adds an extension property to the Message Queuing message, if it does not already exist. Within the extension property, MSMQ-MQSeries Bridge creates an extension field containing the following data:

- The globally unique identifier (GUID) subfield contains a MSMQ-MQSeries Bridge error GUID (different from the GUID used for MQMD).
- The length subfield contains the value 4.
- The data subfield contains a reason code, identical to the codes returned by the MQSeries function MQPUT.

The MSMQ-MQSeries Bridge error GUID is the value of the sg_MSMQExtReasonCode constant, which is defined in the Mqsrext.h include file found in the SDK\Include subdirectory.

If wanted, a Message Queuing application can read messages from the dead letter queue and interpret the reason codes.

Other Extension Fields

If you send an MQSeries message having certain values of MQMD.Report, the MSMQ-MQSeries Bridge adds a second extension field to the new Message Queuing (also known as MSMQ) message. This field is for internal use only, not for use in your applications. The MSMQ-MQSeries Bridge distinguishes the Report extension field from the MQMD and error extension fields by labeling them with different globally unique identifiers (GUIDs).

For more information about other extension fields, see Converting Messages Sent from MQSeries to Message Queuing.

How MSMQ-MQSeries Bridge Converts a Message Extension

If you send a Message Queuing (also known as MSMQ) message including a message extension to MQSeries, the MSMQ-MQSeries Bridge converts the extension in the following way:

- The MSMQ-MQSeries Bridge looks for an extension field identified by the MSMQ-MQSeries Bridge GUID. If it finds one, it reads the MQMD structure from the extension field.
- MSMQ-MQSeries Bridge ignores any other extension fields that may be present in the message extension.
- MSMQ-MQSeries Bridge includes the MQMD structure that it reads from the extension field in the new MQSeries message.

The MQMD structure in the message extension overrides the default MQMD conversions, which are described in Converting Messages Sent from Message Queuing to MQSeries.

A few exceptions to these rules are described in the following sections.

In This Section

- Sender and User Identifiers
- Version Identifiers

Sender and User Identifiers

MSMQ-MQSeries Bridge reads the value of **MQMD.UserIdentifier** stored in the message extension and assigns the value to the **MQMD.UserIdentifier** field in the new MQSeries message.

Version Identifiers

MQSeries uses the following fields for version identification:

- **MQMD.StrucId**
- **MQMD.Version**

When the MSMQ-MQSeries Bridge converts a message extension, it confirms that the values of these fields are for a version of MQSeries that the Bridge supports. If they are not, MSMQ-MQSeries Bridge places the message on the dead letter queue and does not transmit it to MQSeries.

For more information about the supported versions of MQSeries, see Platforms Supported by MSMQ-MQSeries Bridge Extensions. For more information about the permitted values of the version identification fields, see Converting Messages Sent from Message Queuing to MQSeries.

Using Message Extensions

This section suggests a few ways that you can use the message extension property in your messaging applications.

In This Section

- Sending an MQSeries Message to Message Queuing
- Sending a Message Queuing Message to MQSeries

Sending an MQSeries Message to Message Queuing

MSMQ-MQSeries Bridge stores the complete MQMD structure of an MQSeries message in a Message Queuing (also known as MSMQ) message extension. A Message Queuing application can read the extension and retrieve the original MQMD structure.

In this way, a Message Queuing application can retrieve the original values of every MQMD field, regardless of the MSMQ-MQSeries Bridge conversions.

Sending a Message Queuing Message to MQSeries

When you send a Message Queuing (also known as MSMQ) message to MQSeries, you can include a message extension. This can be an extension that you originally received from MQSeries, or one that you created yourself. MSMQ-MQSeries Bridge reads the message extension and uses it to set the MQMD fields of the converted message that it sends to MQSeries.

You can use this feature for two purposes:

- To override the default conversions described in the section
  Converting Messages Sent from Message Queuing to MQSeries

- To supplement the default conversions by assigning MQMD fields that have no Message Queuing equivalent.

The following are some examples of fields that have no equivalents or only partial equivalents (different permitted values or length) among the Message Queuing message properties:

- **MQMD.AccountingToken**
- **MQMD.ApplOriginData**
- **MQMD.CorrelId**
- **MQMD.MsgId**
- **MQMD.MsgType**
- **MQMD.PutApplName**
- **MQMD.PutApplType**
- **MQMD.ReplyToQ**
- **MQMD.ReplyToQMgr**
- **MQMD.Report**

Suppose you want to send a message to an MQSeries application including an MQMD.MsgType value of MQMD_REPLY. The default message conversions provide no way to set this particular value. You can send the value by storing an MQMD data structure in a message extension.

Programming a Message Extension

In a Message Queuing (also known as MSMQ) application, you can program a message extension containing an arbitrary number of extension fields. For use with MSMQ-MQSeries Bridge, build the extension according to the following specifications.

It is recommended that you use MSMQ-MQSeries Bridge Extension Property API to construct the extension with the required syntax. Create a **PROPID_M_EXTENSION** property containing at least one extension field.

Store the MSMQ-MQSeries Bridge globally unique identifier (GUID) code in the GUID subfield of exactly one extension field. The GUID is the value of the sg_MSMQExtMQMD constant, which is defined in the Mqsrext.h include file of the Extension Property API. Store the size of MQMD in the length subfield. Copy a complete MQMD structure byte-for-byte into the data buffer subfield.

When you send the message to MQSeries, the MSMQ-MQSeries Bridge converts the extension field identified by the MSMQ-MQSeries Bridge GUID. Any other extension fields are ignored.

MSMQ-MQSeries Bridge Extension Property API

The MSMQ-MQSeries Bridge Extension Property API is recommended for programming and working with message extensions. The API provides a library of functions that help you create and interpret message extensions. The MSMQ-MQSeries Bridge Extension Property API is supplied as part of the Host Integration Server SDK.

Your Message Queuing (also known as MSMQ) applications can use the API to do the following:

- Read message extensions that you receive from MQSeries
- Create or modify message extensions that you send to MQSeries
- Read or create message extensions for any other Message Queuing messaging purpose

The MSMQ-MQSeries Bridge Extension Property API enables you to create and work with the Message Queuing message extension property. This section provides a few guidelines for using the API functions.

The MSMQ-MQSeries Bridge Extension Property API operates directly on the EP representation of a message extension. In particular, the API functions support the following programming approach:

- Creating or deleting an **EP** object.
- Creating, finding, reading, writing, or deleting extension fields in an **EP** object.
- Converting an **EP** object to a message extension. (**PROPID_M_EXTENSION**) that you can send in a Message Queuing message.
- Converting a message extension that you received in a Message Queuing message to an **EP** object.

You cannot send an **EP** object directly in a Message Queuing message. You must first convert it to a message extension (**PROPID_M_EXTENSION**).

To use the message extension API, include the MSMQext.h header file in your applications. This header files contains constants and function prototypes for the Extension Property API functions. This header file is located in the SDK\Include directory on the Host Integration Server CD and is installed when the SDK package is selected.

If you are using the API in conjunction with MSMQ-MQSeries Bridge, also include the Mqsrext.h header file located in the SDK\Include directory. This file defines the GUID that labels the extension fields.

The Extension Property API functions are handle-based. The following is a summary of the MSMQ-MQSeries Bridge Extension Property API functions. For complete details, see the individual function descriptions in MSMQ-MQSeries Bridge Extensions Reference.

Function	Description
EPAdd	Adds a new extension field to an **EP** object.
EPClose	Frees the extension handle and associated memory of an **EP** object.
EPDelete	Deletes an extension field from an **EP** object.
EPDeleteAll	Deletes all extension fields or all extensions fields matching a specific GUID from an **EP** object.
EPGet	Positions to and optionally retrieves a requested extension field from an **EP** object, storing the GUID, length, and d ata subfields in separate variables. **EPGet** can also be used to locate extension fields containing a specified GUID.
EPGetBuffer	Converts an **EP** object to a message extension and packs the message extension into the supplied buffer.
EPOpen	Creates an **EP** object and optionally unpacks the supplied message extension buffer into it.
EPUpdate	Writes new data to an existing extension field of an **EP** object.

Programming Considerations When Using MSMQ-MQSeries Bridge Extensions

The limitations of specific API functions result from the fact that MSMQ-MQSeries Bridge transmits messages, not API calls, between queuing systems. The MSMQ-MQSeries Bridge transmits messages across the MSMQ-MQSeries interface. The MSMQ-MQSeries Bridge does not transmit API calls across the interface. Thus, Microsoft® Message Queuing (also known as MSMQ) API calls operate only within the Message Queuing environment, and MQSeries API calls operate only within the MQSeries environment. This principle limits the ways in which you can create and access queues.

All Message Queuing API functions operate only within the Message Queuing environment, up to and including foreign computers and queues. For example, you can use the following functions:

- **MQLocateBegin**, **MQLocateNext**, and **MQLocateEnd** to search for foreign queues
- **MQGetMachineProperties**, **MQGetPrivateComputerInformation**, **MQGetQueueProperties**, and **MQGetQueueSecurity** functions to retrieve the properties of foreign queues
- **MQOpenQueue** to open a foreign queue
- **MQSetQueueProperties** and **MQSetQueueSecurity** to set the properties of foreign queues
- **MQCloseQueue** to close a foreign queue

When creating a queue, you can call the Message Queuing function **MQCreateQueue** to create a foreign queue representing an MQSeries queue, but you cannot create the actual MQSeries queue itself. Similarly, you cannot create a Message Queuing queue by calling the MQSeries function **MQOPEN**.

To communicate across the MSMQ-MQSeries interface, your Message Queuing and MQSeries applications should each create their own queues. Alternatively, you can use administration tools such as the Message Queuing Manager or the MQSeries command interface to create the queues.

For proper message delivery, you must ensure that the destination queue for each message actually exists.

You can use the Message Queuing **MQPathNameToFormatName** function to determine a Message Queuing format name for an MQSeries queue. The format name actually refers to the Message Queuing foreign queue. The MSMQ-MQSeries Bridge processes the format name that it finds in a message and directs the message to the MQSeries queue.

When opening a queue, a call to the Message Queuing function **MQOpenQueue** opens the foreign queue, not the MQSeries queue itself. The MSMQ-MQSeries Bridge opens the MQSeries queue as necessary when it transmits a message. If you are sending messages to more that one MQSeries queue, you must open each one separately using its own Message Queuing format name.

In the opposite direction, the MQSeries function **MQOPEN** opens the transmission queue for the Message Queuing computer. The MSMQ-MQSeries Bridge opens the Message Queuing queue when it transmits a message.

When sending a Message Queuing message to a foreign queue with MQSendMessage, Message Queuing delivers the message to the connector queue in the MSMQ-MQSeries Bridge computer. The MSMQ-MQSeries Bridge converts and transmits the message to MQSeries queue. MQSeries delivers a message sent by **MQPUT** to a transmission queue. The MSMQ-MQSeries Bridge reads the message from the MQSeries transmission queue. After converting the message from MQSeries to Message Queuing message properties, MSMQ-MQSeries Bridge transmits the message to the destination Message Queuing queue.

When receiving a message, the MSMQ-MQSeries Bridge does not transmit receive requests across the MSMQ-MQSeries interface. A Message Queuing application can receive a message only from a native Message Queuing queue (the **MQReceiveMessage** function). An MQSeries application can receive only from a native MQSeries queue (the **MQGET** function).

When sending a message from MQSeries to Message Queuing, if you want the MQSeries message to have a value for MQMD.ApplIdentityData, you need the set both of the following:

- Set the **Open** option with **MQOO_SET_IDENTITY_CONTEXT**

- Set the **Put** option with **MQPMO_SET_IDENTITY_CONTEXT**

When a message is retrieved from Message Queuing, the MSMQ-MQSeries Bridge will have converted the Message Queuing **PROPID_M_LABEL** and **PROPID_M_LABEL_LEN** properties from the MQSeries MQMD.ApplIdentityData field value.

The MQMDE extension has the following GUID for MQMD version 2.

```
static const GUID sg_MSMQExtMQMDE =
{ 0x18ae68f5, 0x989b, 0x11d3,
    { 0x8d, 0xf9, 0x0, 0x0, 0xf8, 0x1a,  0xea, 0x1f }
};
```

In This Section

- Transaction Support Using MSMQ-MQSeries Bridge
- Security Using MSMQ-MQSeries Bridge
- Troubleshooting MSMQ-MQSeries Bridge Extensions

Transaction Support Using MSMQ-MQSeries Bridge

The MSMQ-MQSeries Bridge supports both Message Queuing (also known as MSMQ) and MQSeries transactions. The procedure for sending a group of transacted messages is similar in either direction, from Message Queuing to MQSeries or from MQSeries to Message Queuing. Your application should follow these three basic procedures:

- Open a transaction
- Send the messages
- Commit the transaction

At this point, the group of messages reaches the Message Queuing connector queue or the MQSeries transmission queue. Only then does the MSMQ-MQSeries Bridge transmit the messages to the other messaging system. If your application stops the transaction instead of committing, MSMQ-MQSeries Bridge does not handle the messages at all.

Even after you commit a transaction, it is still possible that MSMQ-MQSeries Bridge cannot transmit all the messages. This may occur, for example, if some of the messages are addressed to queues that do not exist in the recipient messaging system. MSMQ-MQSeries Bridge places any undeliverable messages on its dead letter queue.

In the Message Queuing-to-MQSeries direction, the MSMQ-MQSeries Bridge sends transacted messages by a transactional message pipe and untransacted messages by regular message pipe. In the MQSeries-to-Message Queuing direction, MQS->Message Queuing message pipe, and MQS->Message Queuing transactional message pipe do not depend on transaction status.

Security Using MSMQ-MQSeries Bridge

Message authentication and message body encryption are supported from the Message Queuing (also known as MSMQ) sending application up to MSMQ-MQSeries Bridge. Authentication and message body encryption from MSMQ-MQSeries Bridge to MQSeries, or from MQSeries to Message Queuing, are not currently supported.

Troubleshooting MSMQ-MQSeries Bridge Extensions

The MSMQ-MQSeries Bridge generally ignores warnings that it receives from Message Queuing (also known as MSMQ) or MQSeries, but errors are not ignored. Where possible, the MSMQ-MQSeries Bridge transmits messages despite any warnings.

If the MSMQ-MQSeries Bridge is unable to transmit a message to Message Queuing or MQSeries, it places the message on one of its dead letter queues. This can happen, for example, if a message contains an unsupported Message Queuing or MQSeries version identifier or if MSMQ-MQSeries Bridge encounters an error in the recipient messaging system.

The dead letter queues are Message Queuing queues located on the MSMQ-MQSeries Bridge computer. There are two dead letter queues used for this purpose with the following names.

Dead letter queue names	**Comments**
MQBridge dead letter	Used for untransacted messages when errors occur.
MQBridge xact dead letter	Used for transacted messages when errors occur.

Note that these are different from the Message Queuing and MQSeries dead letter queues, where the messaging systems place expired or incorrectly addressed messages.

You can determine whether there are messages on the dead letter queues using the MSMQ-MQSeries Bridge Manager.

If the MSMQ-MQSeries Bridge cannot deliver a message to MQSeries, it records the error in the extension property **PROPID_M_EXTENSION** of the original Message Queuing message and places the message on the dead letter queue. This extension property can be examined to determine the nature of the error encountered.

Registry Settings Used By MSMQ-MQSeries Bridge Extensions

The MSMQ-MQSeries Bridge Extensions uses a number of registry settings for configuration and proper operation. The configuration registry settings are located under the **HKEY_LOCAL_MACHINE\Software\Microsoft\SNA Server\CurrentVersion\Setup** key. These registry settings include the subkeys listed in the following table.

Subkey	Comment
Root Dir	Stores the path to the root directory where the Host Integration Server was installed. The system directory below this root directory is the location where the MSMQ-MQSeries Bridge Extensions dynamic link library (DLL) and other support DLLs are installed.

Creating a Single Sign-On Application

This section provides information about Enterprise Single Sign-On (SSO) technology.

In This Section

- Programming Single Sign-On Overview
- Programming with Enterprise Single Sign-On

Programming Single Sign-On Overview

A business process that relies on several different applications is likely to face the challenge of dealing with several different security domains. Accessing an application on a Microsoft Windows operating system might require one set of security credentials, whereas accessing an application on an IBM mainframe might require different credentials. Dealing with this profusion of credentials is hard for users, and it can pose an even greater challenge for automating processes. To address this problem, BizTalk Server includes Enterprise Single Sign-On (SSO). SSO lets you map a Windows user ID to non-Windows user credentials. This service can simplify business processes that use applications on diverse systems.

To use SSO, an administrator defines affiliated applications, each of which represents a non-Windows system or application. An affiliated application might be a Customer Information Control System (CICS) application that is running on an IBM mainframe, an SAP ERP system that is running on UNIX, or any other kind of software. Each of these applications has its own mechanism for authentication, and so each requires its own unique credentials.

SSO stores an encrypted mapping between the Windows user ID of a user and the associated credentials for one or more affiliated applications. These linked pairs are stored in an SSO database. SSO uses the SSO database in two ways. The first way, called *Windows-initiated Single Sign-On*, uses the user ID to determine to which affiliated applications the user has access. For example, a Windows user account might be linked with credentials that grant access to a DB2 database running on a remote AS/400 server. The second way, called *host-initiated Single Sign-On*, acts in reverse: determining what remote applications have access to a specified user ID, and the privileges that go with that account. For example, a remote application might be linked with credentials that grant access to a user account that has administration privileges on a Windows network.

Note that SSO also includes administration tools to perform various operations. All operations performed on the SSO database are audited; for example, tools are provided that enable an administrator to monitor these operations and set various audit levels. Other tools enable an administrator to disable a particular affiliated application, turn on and off an individual mapping for a user, and perform other functions. There is also a client program that enables end users to configure their own credentials and mappings.

One of the administrative requirements for Single Sign-On is that your local system must know about the credentials that are required to log on to a remote system. Similarly, the remote system must know about the credentials on your local system. Therefore, when you update your credentials, such as when you update your password on your local computer, you must also inform the remote systems that you have done so. The component you design that synchronizes passwords across an enterprise is called a *password sync adapter*.

In This Section

Single Sign-On Interface

The following table describes the COM and .NET Framework interfaces that make up the Single Sign-On interface.

.NET	COM	Description
Microsoft.EnterpriseSingleSignOn.Interop.ISSOAdmin	ISSOAdmin Interface (COM)	Creates, updates, and deletes an SSO application. Also performs other administration functions.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOConfigStore	ISSOConfigStore Interface (COM)	Gets and sets information in the SSO configuration store.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOLookup1	ISSOLookup1 Interface (COM)	Enables you to look up the external credentials on a specified application for the current user.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOLookup2	ISSOLookup2 Interface (COM)	As above, but also enables you to look up the Windows credentials for a specified external user.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOMapper	ISSOMapper Interface (COM)	Enables you to set the external credentials for the current user for a specified application.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOMapping	ISSOMapping Interface (COM)	Creates and maintains the mapping between users and affiliated applications.
Microsoft.EnterpriseSingleSignOn.Interop.ISSOTicket	ISSOTicket Interface (COM)	Creates the ticket that contains the appropriate security information. This ticket is then sent on with the appropriate message from your application.

In addition, Host Integration Server supports the Password Sync Helper (PS Helper) component. You can use PS Helper when you design password synchronization components. The following table describes the COM and .NET Framework interfaces exposed by PS Helper.

.NET	COM	Description
Microsoft.EnterpriseSingleSignOn.Interop.ISSOPSWrapper	ISSONotification Interface (COM)	Handles password changes to and from non-Windows operating systems.
	SExternalAccount Structure (COM)	Describes an external account.
	SPasswordChange Structure (COM)	Describes a password change.
	SPasswordChangeComplete Structure (COM)	Describes the completion of a password change.
	SStatus Structure (COM)	Describes an error or event.
	SAdapterInGroup Structure (COM)	Describes the adapters in a given group.
	SAdapter Structure (COM)	Describes a specific adapter.

Single Sign-On Applications

From a programming perspective, you can write two different kinds of applications using Single Sign-On: a traditional Single Sign-On application that uses the Single Sign-On interface to interact with remote applications, and a password sync adapter that uses the Password Sync (PS) Helper interface to synchronize passwords across your enterprise.

In This Section

Traditional Single Sign-On Applications

The Single Sign-On (SSO) programming architecture contains a mapping component to map between applications and users, a lookup component to look up credentials for a specified use, and an administration component to perform administrative tasks. In addition, SSO also contains a ticketing interface so that your application can issue and redeem tickets.

Mapping

Mapping is the process of linking a specified user with a specified application. You can map between affiliate applications and users who are using **ISSOMapping**, **ISSOMapper**, and **ISSOMapper2**. With **ISSOMapping**, you can create, delete, enable, and disable mappings. With **ISSOMapper**, and **ISSOMapper2**, you can get and set mapping data for the current user.

Lookup

The core of the Single Sign-On programming interface is the ability to look up credentials for specified users. You can look up credentials using **ISSOLookup1**, and **ISSOLookup2**. **ISSOLookup1**, enables the user to retrieve their own external credentials. In contrast, **ISSOLookup2** also enables you to look up the credentials of a remote user for a local affiliate application. The former is necessary for a traditional Single Sign-On application, whereas the latter is useful when you are writing a host-initiated Single Sign-On application.

Administration

You can perform many of the administrative capabilities programmatically through **ISSOAdmin**, **ISSOAdmin2**, and **ISSOConfigStore**. Such tasks include configuring Single Sign-On and creating and describing an application to Single Sign-On. You also can create and modify SSO databases using **ISSOConfigDB**, **ISSOConfigOM**, and ISSOConfigSS. Finally, you can administer the password sync features using **ISSOPSAdmin**.

Communication and Ticketing

Your application will most likely issue messages so that your user can communicate with a remote application. To communicate with a remote application more securely, you must issue and redeem a Single Sign-On ticket. You can also issue and redeem tickets programmatically.

See Also

Other Resources

Single Sign-On Applications

Password Sync Adapters

A *password sync adapter* is a component that propagates password changes to and from a non-Windows system. Although password sync adapters are similar to traditional Single Sign-On applications, they have several differences:

- They are administered using a specialized interface.
- They are described using a specialized XML format in the configuration store.
- Host Integration Server uses a specialized feature to organize adapters in the configuration store.

In This Section

Password Sync Programming Architecture

A password sync adapter uses a pull model for interacting with the rest of the Enterprise Single Sign-On system: that is, the adapter actively receives password changes from the Enterprise Single Sign-On (ENTSSO) service and also from the non-Windows system. Similarly, the adapter pushes password changes received from one system to the other. With this model, your adapter interacts with three architectural components: the ENTSSO architecture, the Password Sync (PS) Helper component, and a specified non-Windows system.

Enterprise Single Sign-On Architecture

ENTSSO is the service that implements the Enterprise Single Sign-On technology, and runs as a local service on the same system as your adapter. Therefore, communication between the adapter and ENTSSO is always local. However, a password sync adapter runs in a separate process from the ENTSSO service.

ENTSSO uses the configuration store to store configuration information for an adapter. The Application Users account of a configuration store application corresponds to the access account. When an adapter calls into ENTSSO, ENTSSO checks that the adapter is within the configured access account for that adapter. The access account must be a (local or domain) group account.

Because ENTSSO stores information about an adapter, the adapter can identify itself to ENTSSO by its adapter name. The adapter name corresponds to the configuration store application name and the configuration store identifier used to store the adapter properties. Therefore, adapters must know only their adapter name to access configuration information and to correctly identify themselves to the ENTSSO system at run time.

Password Sync Helper

Password Sync (PS) Helper is a COM component provided by ENTSSO. PS Helper runs in-process with the password sync adapter, and exposes ISSOPSWrapper. Your adapter can call either interface to communicate with the ENTSSO service. The PS Helper passes communications to and from ENTSSO using encrypted (packet privacy) lightweight remote procedure call (LRPC). Although the communications are mainly for password updates, you can also use the interfaces to pass other types of notifications between the adapter and ENTSSO. Because PS Helper runs as a singleton value per process, it is possible for multiple adapters to call the same PS Helper object from within the same adapter process. For more information about using PS Helper programmatically, see Synchronizing Passwords.

Non-Windows System

The non-Windows system is the remote computer your adapter interacts with. How you interact with the non-Windows system is up to you.

See Also

Other Resources

Password Sync Adapters

Adapter Programming Administration

An adapter is a special type of configuration store application: that is, an adapter is a component that shares a namespace with other Single Sign-On and configuration store applications. Therefore, you can access information about an adapter using ISSOConfigStore. But unlike a configuration store application, you do not perform administrative functions on an adapter with the ISSOAdmin interface. Instead, you administer an adapter through ISSOPSAdmin. The reason for a specialized adapter administration interface is so that the system can coordinate other activities with the configuration store.

See Also

Concepts

Adapter Programming Configuration

Adapter Groups and Group Adapters

Other Resources

Password Sync Adapters

Adapter Programming Configuration

Every type of password sync adapter has different configuration requirements, depending on what non-Windows system you design the adapter for. Like affiliate applications, you can use the Enterprise Single Sign-On configuration store to store configuration properties centrally and more securely.

As with other configuration store applications, an administrator can use the Enterprise Single Sign-On management user interface to locate and read an XML file that describes the configuration properties for your adapter. The management tools use the XML file to render a property page to gather the required property values, for the specified adapter. You can also use ISSOConfigStore to load and read XML name/value combinations to and from the configuration store, or you can use the SSOPS tool.

You can also use the Enterprise Single Sign-On administration tools to enable and disable an adapter. On initial creation, an adapter is disabled.

See Also

Other Resources

Password Sync Adapters

Adapter Groups and Group Adapters

An *adapter group* is an administration mechanism that you can use to collect and organize a set of adapters. In contrast, a *group adapter* is a component that services all adapters in an adapter group. For example, you might write a set of adapters that all use the same COM component to transmit password synchronizations over TCP/IP. Your set of adapters is called the adapter group, whereas the component that services them all is called a group adapter. Adapter groups are described in the configuration store. You can retrieve information and updates on an adapter group by using **ISSOPSAdapter.ReceiveNotification**.

A group adapter has the same name as the adapter group. Other than the naming restriction, a group adapter is otherwise identical to a normal adapter. For example, a group adapter can have independent access groups and configuration properties, as described by its configuration file. A group adapter is most likely on the same computer as any adapters in the adapter group. However, this is not currently enforced. Likewise, all adapters in the same adapter group can be expected to be on the same computer.

By using **ISSOPSAdapter.InitializeAdapter**, you can access and initialize a group adapter during startup. When you initialize a group adapter, the system informs the group adapter of all adapters in the adapter group on the current system. In addition, the system sends notifications to the group adapter any time an adapter is added, deleted, enabled, or disabled in the adapter group. However, the group adapter does not receive any password change notifications.

Adapter groups and group adapters are optional using the Administration tool.

See Also
Other Resources
Password Sync Adapters

What You Should Know Before Programming Single Sign-On

To use this documentation effectively, you should be familiar with the following:

- Microsoft Windows XP, Windows 2000 Server, Windows Server 2003, or Windows Server 2008 operating systems. You should be especially familiar with Windows Security features.
- Administrative features of Enterprise Single Sign-On, especially how to perform administrative actions using the user interface.

Depending on the API and development that you are using, you should also be familiar with the following interfaces:

- COM
- The .NET Framework and the common language runtime
- Windows Networking, and specifically how to send and receive tickets

See Also

Other Resources

Programming Single Sign-On Overview

Supported Platforms for Single Sign-On

Enterprise Single Sign-On is included with BizTalk Server. Therefore, it is supported by all operating systems that support BizTalk Server.

See Also

Other Resources

Programming Single Sign-On Overview

Programming with Enterprise Single Sign-On

Enterprise Single Sign-On (SSO) is a technology implemented in the Enterprise Single Sign-On service (ENTSSO) that enables users to access multiple remote services without having to work their way through additional sign-on screens.

In This Section

How to Determine Current Single Sign-On Access

One of the first tasks you might need to perform for a user is to determine what affiliated applications have already been set up for the current user. You can perform this query with a call to ISSOMapper.GetApplications.

To query the Single Sign-On database for the applications available to the current user

1. Create a new instance of **ISSOMapper**.

   In general, **ISSOMapper** is an interface designed to retrieve information from Single Sign-On (SSO). You will most likely use **ISSOMapper** in many similar queries.

2. Retrieve all applications that are affiliated with the current user by calling GetApplications.

   GetApplications automatically returns only the affiliated applications of the current user.

The following code example demonstrates how to query the Single Sign-On database.

```
private static string[] Applications=null;
. . .
public static string[] GetCurrentUserApplications()
{
   if(Applications==null)
   {
      string[] descs;
      string[] contacts;
      ISSOMapper mapper=new ISSOMapper();
      mapper.GetApplications(out Applications, out descs, out contacts);
   }
   return Applications;
}
```

See Also

Other Resources

Programming with Enterprise Single Sign-On

How to Configure Single Sign-On

Before accessing Enterprise Single Sign-On, you should make sure that Enterprise Single Sign-On is set correctly for the current user. For most configurations, you use one of two interfaces. **ISSOAdmin** is the general administration interface that enables you to create new affiliation applications. However, by using ISSOAdmin.GetGlobalInfo and ISSOAdmin.UpdateGlobalInfo, you can set a variety of flags and administration values. One possible task, as described in the following procedure, is to ensure that SSO ticketing has been enabled.

To enable ticketing

1. Create a new instance of **ISSOAdmin**.
2. Retrieve the current settings through **ISSOAdmin.GetGlobalInfo**.

   If necessary, you may want to confirm that the flags are set to the correct values at this point.
3. Change any relevant flags using **ISSOAdmin.UpdateGlobalInfo**.

   In this particular case, all the flags are being set to validate and enable tickets.

The following example shows how to enable ticketing using Single Sign-On.

```
public static bool EnableTickets()
{
   try
   {
      ISSOAdmin admin=new ISSOAdmin();
      int flags=0;
      int appDeleteMax=1000;
      int mappingDeleteMax=1000;
      int ntpLookupMax=-1000;
      int xplLookupMax=-1000;
      int ticketTimeout=2;
      int cacheTimeout=60;
      string secretServer=null;
      string ssoAdminGroup=null;
      string affiliateAppMgrGroup=null;
      // Get current default settings.
      admin.GetGlobalInfo(out flags, out appDeleteMax, out mappingDeleteMax, out ntpLookupM
ax, out xplLookupMax, out ticketTimeout, out cacheTimeout, out secretServer, out ssoAdminGr
oup, out affiliateAppMgrGroup);
      // Update global settings.
      admin.UpdateGlobalInfo(SSOFlag.SSO_FLAG_ALLOW_TICKETS | SSOFlag.SSO_FLAG_VALIDATE_TIC
KETS, SSOFlag.SSO_FLAG_ALLOW_TICKETS | SSOFlag.SSO_FLAG_VALIDATE_TICKETS, ref appDeleteMax,
ref mappingDeleteMax, ref ntpLookupMax, ref xplLookupMax, ref ticketTimeout, ref cacheTimeo
ut, null, null, null);
   }
   catch
   {
      return false;
   }
return true;
}
```

See Also

Other Resources

Programming with Enterprise Single Sign-On

How to Create and Describe an Application to Single Sign-On

A common administrative task that you might need to perform is adding an affiliate application into the Enterprise Single Sign-On (SSO) database. Adding an affiliate application to the Enterprise SSO database enables you to associate users and credentials with the affiliated application.

Note
Creating an affiliated application requires membership in the "SSO Affiliate Administrator" account or above.

To create and describe an application in the SSO database

1. Create a new **ISSOAdmin** object.
2. Create a new application with a call to **ISSOAdmin.CreateApplication**.
3. Add the relevant fields describing the application with a call to **ISSOAdmin.CreateFieldInfo**.

   During this step, you tell the database that an application has users and associated passwords.
4. Push the newly created description out to the server with a call to **ISSOAdmin.UpdateApplication** or **ISSOAdmin2.UpdateApplication2**.

   The difference between the two methods is that **UpdateApplication2** uses an **IPropertyBag** as the way to describe the application updates, while **UpdateApplication** has multiple parameters.
5. Purge the local cache for the changes you made by calling **ISSOAdmin.PurgeCacheForApplication**.

   Purging the local cache is a security measure that prevents having the names and passwords that you describe in step 3 to exist in an unsecured location.

The following example shows how to create an application and add field information.

```
   public static bool AddApplication(string name, string admins, string users)
       {
          try
          {
             ISSOAdmin admin=new ISSOAdmin();
             // Create application.
             admin.CreateApplication(name, "SSO Sample Application", "administrator@ssoaffil
iateapplication.com", users, admins, SSOFlag.SSO_WINDOWS_TO_EXTERNAL | SSOFlag.SSO_FLAG_ALL
OW_TICKETS | SSOFlag.SSO_FLAG_VALIDATE_TICKETS, 2);
             // Add fields.
             admin.CreateFieldInfo(name, "User Id", SSOFlag.SSO_FLAG_NONE);
             admin.CreateFieldInfo(name, "Password", SSOFlag.SSO_FLAG_FIELD_INFO_MASK);
             // Enable application.
             admin.UpdateApplication(name, null, null, null, null, SSOFlag.SSO_FLAG_ENABLED,
SSOFlag.SSO_FLAG_ENABLED);
             // Purge changes.
             admin.PurgeCacheForApplication(name);
          }
          catch
          {
             return false;
          }
          return true;
       }
```

See Also

Other Resources

Programming with Enterprise Single Sign-On

How to Map Single Sign-On Credentials

When you know that you have affiliated applications in your Enterprise Single Sign-On database, you can map the credentials for a user to that application. Mapping the credentials of the current user to an affiliated application requires that you use a combination of the **ISSOMapper** and **ISSOMapping** interfaces.

To map between an affiliated application and user credentials

1. Create new instances of **ISSOMapper** and **ISSOMapping**.
2. Set the **ISSOMapping** properties to the relevant values.

   The relevant properties for **ISSOMapping** are the Microsoft Windows domain name of the user, the Windows user name, the name of the affiliated application, and the external user name.
3. Create the mapping with a call to ISSOMapping.Create.

   Calling **ISSOMapping.Create** propagates the local copy of the mapping out to the Enterprise Single Sign-On server.
4. Set the credentials on the mapping with a call to **ISSOMapper.SetExternalCredentials**.
5. Enable the mapping with a call to **ISSOMapping.Enable**.

The following example shows how to add mapping between a specified Enterprise Single Sign-On application and a user.

```
public static bool AddMapping(string application, string user, string XU, string XP)
{
   try
   {
      // Set mapping.
      ISSOMapper mapper=new ISSOMapper();
      ISSOMapping mapping=new ISSOMapping();
     string username=user.Substring(user.IndexOf('\\')+1);
      string userdomain=user.Substring(0, user.IndexOf('\\'));
      mapping.WindowsDomainName=userdomain;
      mapping.WindowsUserName=username;
      mapping.ApplicationName=application;
      mapping.ExternalUserName=XU;
      mapping.Create(0);
      // Set credentials.
      string[] credentials=new string[]{XP};
      mapper.SetExternalCredentials(application, XU, ref credentials);
      mapping.Enable(0);
   }
   catch
   {
      return false;
   }
   return true;
      }
```

See Also

Other Resources

Programming with Enterprise Single Sign-On

Logging on to a Remote or Local Application

After you finish setting affiliate credentials for your user, you can use Enterprise Single Sign-On service (ENTSSO) to provide access to applications. If the user is local, you can use ENTSSO to retrieve credentials for a non-Windows application. In contrast, if the user is remote, you can use ENTSSO to retrieve credentials for a local application.

In This Section

How to Log a Local User on to a Non-Windows Application

After you set up your user with an affiliate application, you can use Single Sign-On (SSO) to access the external user name and credentials of the current user. Using these credentials, you can then log your user on to the affiliate application that is running on a host server.

Note
In addition to setting the appropriate security protocols for SSO, you might also need to set additional security to allow your application to call SSO in the correct security context. If your application cannot call SSO in the correct security context, SSO will deny access to your application.

To set the security context for an SSO application

1. Identify what credentials your application needs to run successfully.

   For example, an application that uses Web services or.NET Framework remoting hosted in IIS needs to impersonate the client in order to pass the appropriate credentials on to SSO.

2. Confirm that the relevant security settings, such as those on virtual directories, application pools, and web.config files, are set to provide your application with those credentials.

   For more information about how to set security credentials, see Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication.

   For more information about passing credentials for an ASP.NET Web service, see HOW TO: Pass Current Credentials to an ASP.NET Web Service.

To log a local user on to a host application

1. Receive the request to log the current user on to an application running on the host server.

   It is your responsibility to determine how the current user requests to be logged on to a host application.

2. Retrieve the credentials for the current user who is using **ISSOLookup1.GetCredentials** or **ISSOLookup2.GetCredentials**.

   You must supply the name of the host application together with any relevant flags. **GetCredentials** returns the associated user name and credentials for the host application.

   Note that you can use either **ISSOLookup1** or **ISSOLookup2**. The only difference is that **ISSOLookup2** also has a method for logging a remote user on to a local windows application.

3. Use the external user name and credentials to log on to the host application.

   It is your responsibility to determine how to use the credentials to log on to the host application.

See Also

Tasks

How to Log a Remote User on to a Local Application

How to Log a Remote User on to a Local Application

The other main feature of Enterprise Single Sign-On service (ENTSSO) is supporting a host-initiated process (HIP). ENTSSO interacts with HIP when a remote user tries to access a local Windows resource. Using ENTSSO, you can receive the request from the host user and request access to the local Windows application.

To log a remote user on to a local Windows application

1. Receive the request from the remote user.

   It is your responsibility to determine how to retrieve a request from the remote user.

2. Request that the remote user be given access to the specified affiliate application, using **ISSOLookup2.LogonExternalUser**.

   LogonExternalUser passes in the name of the application the external user wishes to access, the name of the external user, the associated credentials for the external user, and any relevant flags. If successful, **LogonExternalUser** returns a handle to a Windows access token.

   The remote user must already be identified in the Single Sign-On database, have their credentials in the database, and be associated with an affiliate application. Otherwise, **LogonExternalUser** returns an error. You can keep the user names and credentials up to date using Password Sync.

   In addition, you must have protocol transition enabled.

3. Use the Windows handle returned from **LogonExternalUser** to impersonate the user that the token represents.

See Also

Tasks

How to Log a Local User on to a Non-Windows Application

Reference

ISSOLookup2.LogonExternalUser Method

LogonExternalUser

How to Change the Behavior of a Single Sign-On Interface

Many of the objects in the Enterprise Single Sign-On (SSO) object model expose the IPropertyBag interface, which allows you to modify the behavior of the specified object. If you call QueryInterface on an SSO object, you can retrieve the IPropertyBag interface and use it to change the behavior of your current object.

To change the behavior for a specified SSO interface

1. Use QueryInterface on the specified interface to retrieve an IPropertyBag instance.
2. Use IPropertyBag.Write to set the property, type, and value for the interface.

   The following table describes the valid values for the IPropertyBag propName and ptrVar parameters.

propName	Type	ptrValue	Usable On
CurrentSSOServer	VT_BSTR	Name of the server to send the information to	All
Transaction	VT_UNKNOWN VT_EMPTY	A DTC ITransaction pointer, or NULL to clear the scope.	ISSOConfigStore::SetConfigInfo ISSOConfigStore::GetConfigInfo ISSOConfigStore::DeleteConfigInfo ISSOAdmin::CreateApplication ISSOAdmin::DeleteApplication ISSOAdmin::UpdateApplication ISSOAdmin::CreateFieldInfo ISSOMapper::GetFieldInfo
AppFilterFlags	VT_I4 VT_UI4	Flags to control what application to filter.	ISSOMapper::GetApplications ISSOMapper2::GetApplications2
AppFilterFlagsMask	VT_I4 VT_UI4	Flag mask to control what application to filter.	ISSOMapper::GetApplications ISSOMapper2::GetApplications2
AsyncCall	VT_BOOL	True to call using an async RPC; false to use a synchronous RPC.	ISSOConfigOM::GetServerStatus ISSOAdmin::GetGlobalInfo

- **CurrentSSOServer**: the standard behavior for determining which server to send SSO information to is as follows:

  1. Look in the registry for the current user. The server name can be set for the current user using the command line tools or GUI.
  2. Look in the registry for all users. The server name can be set for all users using the command line tools or GUI.
  3. If no SSO server name is found in the registry then use the current computer.

  Setting CurrentSSOServer to a specified server overrides the previous process for the specified interface. Once you set CurrentSSOServer, all subsequent method calls on the interface will be sent to the specified server.

- **Transaction**: specifies a DTC transaction that to scope the operations performed by the SSO object model. You must pass a DTC ITransaction pointer in *ptrValue*, or "null" to clear the current transaction scope.

- **AppFilterFlags/AppFilterMask**: controls what types of applications will be returned from ISSOMapper.GetApplications and ISSOMapper2.GetApplications. If the application flags match the flags specified by the filter flags and the filter flag

mask they will be returned. One way to perform application filtering is to set AppFilterFlagsMask to SSO_FLAG_APP_FILTER_BY_TYPE and to then set AppFilterFlags to one or more of the following:

SSO_APP_TYPE_INDIVIDUAL

SSO_APP_TYPE_GROUP

SSO_APP_TYPE_CONFIG_STORE

SSO_APP_TYPE_HOST_GROUP

SSO_APP_TYPE_PS_ADAPTER

SSO_APP_TYPE_PS_GROUP_ADAPTER

- **AsyncCall**: if true, then SSO will perform the method using an asynchronous remote procedure call (RPC). The method will return E_PENDING while in progress. Any other return value indicates that the method is completed. AsyncCall also allows you to poll the method for completion.

Issuing and Redeeming a Single Sign-On Ticket

After you link a user and an affiliate application, you can issue tickets to help ensure security while maintaining communications. Single Sign-On ticketing works just like other ticketing technologies: before sending the message off, you append the Single Sign-On ticket to the message as a string. The server receives your message, decodes the ticket, and uses the information as appropriate.

To issue a Single Sign-On ticket

1. Create a new ticket object with a call to I**SSOTicket**.
2. Issue the ticket with a call to I**SSOTicket.IssueTicket**.

To redeem a Single Sign-On ticket

1. Create a new ticket object with a call to I**SSOTicket**.
2. Redeem the ticket with a call to I**SSOTicket.RedeemTicket**.

See Also

Other Resources

Programming with Enterprise Single Sign-On

Synchronizing Passwords

You synchronize a password by using a password sync adapter. This adapter should be able to communicate with a specific, remote, non-Windows system, and should also be able to instruct that system to update password information.

In This Section

How to Create a Password Sync Adapter

A password sync (PS) adapter is an application that uses the Password Sync Helper component to pass notifications to and from Enterprise Single Sign-On (SSO). Note that although the PS Helper component exposes a COM and a .NET Framework interface, your adapter does not necessarily have to be a COM component. You can design your adapter as a stand-alone process, a COM+ application, or a Windows service.

To create a password sync adapter

1. Inform Enterprise Single Sign-On service (ENTSSO) that your provider is active using **ISSOPSWrapper.InitializeAdapter**.

   InitializeAdapter informs ENTSSO that a provider, usually the same provider that is making the call, is currently turned on, and therefore will be communicating password updates to and from the system. You can also use **InitializeAdapter** to activate other resources such as group adapters.

2. Send password updates to ENTSSO by using **ISSOPSWrapper.SendNotification**.

   You must determine how you receive password updates from your non-Windows system. After you receive the update, you can pass the information on to ENTSSO using **SendNotification**. Note that **SendNotification** is not limited to sending password updates: the architecture of **SendNotification** also enables you to send other types of notifications.

3. Request password updates from ENTSSO by using **ISSOPSWrapper.ReceiveNotification**.

   Because the password sync adapter is a pull technology, ENTSSO never calls your adapter. Instead, your adapter periodically calls **ReceiveNotification** to see whether any password updates are available. You can choose to set the WAIT flag on **ReceiveNotification**. Setting WAIT blocks the thread until a notification is available.

   Note that ENTSSO delivers a password change to your adapter in plain text. It is the responsibility of the adapter to protect that password information against incorrect disclosure. It is also the responsibility of the adapter to protect itself against spoofing or attacks from other invalid sources, including spoofing of the Password Sync Helper component.

   After you receive a password update from ENTSSO through the *pReceiveNotification* parameter, you must pass this information on to your non-Windows system. As with **SendNotification**, you must determine the best way to communicate with the remote server.

4. Turn off your adapter using **ISSOPSWrapper.ShutdownAdapter**.

   ShutdownApplication should be the last method called by an adapter, and indicates that the adapter will no longer send or receive password updates to ENTSSO.

   Note that ENTSSO buffers any password changes a user makes while the adapter is shut down, up to a buffer size limit.

See Also
Other Resources
Programming with Enterprise Single Sign-On
Synchronizing Passwords

How to Configure a Password Sync Adapter

After you have finished creating your password sync adapter, you must load your adapter on to a system. Additionally, you must inform Enterprise Single Sign-On (ENTSSO) and the configuration store that your application is a password sync adapter. You must register with the configuration store for security purposes: your adapter will request updates to passwords and other credentials. Therefore, ENTSSO must know that a given adapter is allowed to ask for such permissions.

To configure a password sync adapter

1. Create your adapter with the configuration store using the ssops tool.

   Using ssops, you load an XML configuration file into the configuration database that describes your application to Enterprise Single Sign-On.

2. After you create the adapter with the config database, you can modify the adapter information with **ISSOPSAdmin.SetAdapterProperties**.

   While the two methods are similar, **SetAdapterProperties** sends a message to the adapter when the configuration information changes.

3. To delete an adapter, use ISSOAdmin.DeleteApplication.

See Also
Other Resources
Synchronizing Passwords

How to Assign an Application to an Adapter

To process information between a local application and a remote server, an adapter must have one or more applications assigned to it.

To assign an application to an adapter

1. Add the application to the adapter by using **ISSOPSAdmin.AssignApplicationToAdapter**.
2. You can retrieve the current list of applications assigned to an adapter by using **ISSOAdmin.GetApplicationsForAdapter**.
3. Once you are finished, you can remove an application from an adapter by using **ISSOPSAdmin.RemoveApplicationFromAdapter**.

How to Create and Modify an Adapter Group

One of the new features of Single Sign-On (SSO) is the ability to create and modify adapter groups. As the name implies, an adapter group is a collection of adapters. You can use adapter groups to organize security settings and other properties for your adapters.

To create and modify an adapter group

1. Create the adapter group by using a call to the ssops tool.

   In the associated XML file, you must set the group flag as an adapter group.

2. Add one or more adapters to the adapter group by using **ISSOPSAdmin.AddAdapterToAdapterGroup**.

   At this point, your adapter group is ready to work as intended. If necessary, you can view the full list of associated adapters by using **ISSOPSAdmin.GetAdaptersForAdapterGroup**.

3. You can modify the settings of your adapter group using **ISSOPSAdmin.SetAdapterProperties**.

4. When you are finished, you can remove the adapter from the adapter group using **ISSOPSAdmin.RemoveAdapterFromAdapterGroup**.

5. Finally, you can delete the adapter group by using **ISSOAdmin.DeleteApplication**.

   You may choose instead to delete the adapter group by using the **-delete** command of the ssops tool.

See Also
Other Resources
Synchronizing Passwords

Programmer's Reference

This section of the Microsoft Host Integration Server 2009 software development kit (SDK) provides reference information for the programmer writing applications for Host Integration Server 2009.

In This Section

Application Integration Programmer's Reference

Data Integration Programmer's Reference

Network Integration Programmer's Reference

Administration and Management Programmer's Reference

Messaging Programmer's Reference

Security Programmer's Reference

Application Integration Programmer's Reference

This section of the Microsoft Host Integration Server 2009 Programmer 's Reference describes the objects, methods, properties, controls, and other interfaces that enable you to integrate Host Integration Server technologies into your application.

In This Section

Introduction to COM and COM+

Data Types

Host and Automation Data

COMTIContext Interface

COMTIContext Keywords

TI Component Properties

Standard Transaction Request and Reply Messages

CICS Enhanced Listener Request and Reply Messages

Microsoft Concurrent Server

Reference

Application Integration Samples

Related Sections

Application Integration Programmer’s Guide

Introduction to COM and COM+

The Component Object Model (COM) is a binary standard that enables software components to interoperate in a networked environment regardless of the language in which they were developed.

Transaction Integrator (TI) is based on COM, which defines a standard way of handling and reporting errors. (COM is the foundation for COM+.) To understand TI behavior and errors, you must understand how it uses COM.

Objects and clients are the basic building blocks of COM applications. An object is a piece of software that can do a specific task. A client is software that uses and controls objects.

COM can be compared to business relationships. There are many different businesses that provide their customers (clients) with products and services (objects) that the clients can use and control. COM applications are clients that use objects that can do specific tasks. For example, in computer technology, Powerpnt.exe depends on several Microsoft Office and Microsoft Windows utility objects to do most of the work involved with editing and presenting a presentation. The Powerpnt.exe is just a client to those utility objects and directs them to their tasks. When you want to open a .ppt file, a dialog box appears, from which you choose the file to open. This dialog box is displayed by a Windows utility object that can display a **File Open** dialog box, instead of by Powerpnt.exe.

Dividing software into clients and objects enables you to use those clients and objects as building blocks that can be combined and reused in many different ways by many different clients, just as standard building materials can be used to construct many different structures.

COM and COM+ are the key technologies that provide the infrastructure that enables clients and objects to work together. Other key COM concepts include methods, interfaces, classes, references, and components, which build on the foundation of clients and objects.

In This Section

COM Defined

The Component Object Model (COM) is a binary standard that enables software components to interoperate in a networked environment regardless of the language in which they were developed.

See Also

Concepts

COM+ Defined

Component Services Features

Other Resources

Introduction to COM and COM+

COM+ Defined

Component Services (COM+) consists of the latest version of COM, distributed COM (DCOM), and Microsoft Distributed Transaction Coordinator (DTC), plus additional functionality. With COM+, administrators can deploy and administer COM+ applications through a graphical user interface, or automate administrative tasks using a scripting or programming language. Software developers can use COM+ to visually configure routine component and application behavior, such as security and participation in transactions, and to integrate components into COM+ applications.

To open COM+, click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Component Services**. It is also available in Transaction Integrator (TI) Manager: start TI Manager and double-click the **Component Services** folder.

Component Services provides standard application functionality for administration, deployment, security, reliability, and scalability. Component Services simplifies the administrator's work by providing a tool for administering all applications visually through a single user interface.

See Also

Concepts

COM Defined

Other Resources

Introduction to COM and COM+

Component Services Features

Component Services (COM+) builds on COM concepts and adds three major new elements: COM+ applications, increased administrator involvement, and interception. You can gain access to Component Services from within TI Manager.

A COM+ application is a COM component that includes one or more components developed and configured to work together by Component Services to simplify administration of application servers. Each COM+ application unites TI components (type libraries) and COM components with configuration information about how they should be deployed, thus making a single unit for deployment and administration. A wizard helps to deploy COM+ applications. You can easily administer COM+ applications by using Component Services in TI Manager. This feature gives administrators complete control over application server deployment and configuration. COM+ applications also aid in the configuration of an application's presentation tier. Administrators can use Component Services to quickly create application proxies that can be used to configure thousands of client computers.

Component Services expands the role of administrators by empowering administrators to do many tasks that were previously done exclusively by developers. In the application development phase, developers write their own custom type libraries or use TI Designer to create TI component libraries from COBOL transaction programs (TPs). Then developers deploy the TI components in a COM+ application and configure the application settings as needed. The COM+ application is a deployable unit that the developer can pass to the administrator to deploy.

The administrator receives a COM+ application file from a developer, and then uses Component Services to deploy that application on the server computer. Next, the administrator configures permissions to use the application, as well as other settings. Then, the administrator creates an application proxy, which is a file used to configure each client computer in the presentation tier. After using Microsoft Active Directory directory service and IntelliMirror to complete the client configuration, the administrator continues to use Component Services to monitor the application's behavior.

Component Services intercepts all calls from clients to objects, and you can use it to manage objects and their interaction with clients. Because of this interception capability, Component Services governs the interaction between objects and their clients, and does a lot of the work that would be done by objects themselves. You can design objects to do business logic only; Component Services handles everything else by tracking object behaviors and needs and by managing everything the objects do.

The following are the major features of Component Services that are supported by Transaction Integrator (TI):

- **Administration.** You can use the powerful Component Services user interface to administer the application server tier. You can use Component Services to ease deployment, configuration, monitoring, and upgrades. Component Services also gives administrators the ability to use scripts to control all administration.

- **Scalability.** Component Services achieves high scalability by pooling server resources so that when one person is not using them, another person can. When a resource is needed, it is taken from a shared pool. When the resource is no longer needed, it is returned to the pool. Component Services pools many resources, including threads, ODBC connections, OLE DB connections, and COM objects.

Note
TI supports the Microsoft Windows Server 2003 and Windows 2000 COM+ feature known as *object pooling*. To enable object pooling for a TI component, right-click the component, and click **Properties**. Then select the **Enable Object Pooling** check box on the **Activation** tab. There are overall system manageability benefits from enabling this feature for TI components because you can then specify the number of instances to be pre-initialized when the application starts by using the **Minimum Pool Size** field. You can also set an upper limit on the number of active instances by using the **Maximum Pool Size** field to prevent the server running Windows Server 2003 or Windows 2000 from flooding the host system.

- **Transaction Support for Reliability and Data Integrity.** Enterprise applications must have high levels of reliability and data integrity. Component Services provides reliability by offering transaction support. That is, either all the actions in a transaction occur, or none of them occurs. If all of the actions cannot take place, Component Services rolls back all of your object's work so that the databases are left in their original state.

- **Security.** Because Component Services acts as an intermediary between clients and objects, it is able to regulate access to application server functionality. By using the Component Services console, the administrator can give different users different access rights to the application.

- **Queued Components.** Component Services is integrated with another feature of Windows called Message Queuing (also known as MSMQ). Message Queuing gives enterprise applications the ability to continue running even when a server is unavailable. Clients can call COM objects when the COM object is currently unavailable, and the call is stored as a message in Message Queuing. After the server is available again, it retrieves the message from the message queue and processes the call to the COM object.

See Also
Concepts
COM Defined
COM+ Defined
Other Resources
Introduction to COM and COM+

Distributed Applications

Applications that run partly on the Windows operating system and partly on the mainframe are known as distributed applications. Transaction Integrator (TI) components support all local and distributed applications that adhere to the Automation specifications and the distributed COM (DCOM) specifications. DCOM supports communication between objects on different computers on a network by handling the low-level details of network protocols. This enables use of distributed programs that consist of multiple processes working together to accomplish a single task. The Distributed Transaction Coordinator (DTC) is the part of Component Services that coordinates external or two-phase transactions.

See Also

Concepts

COM Defined

COM+ Defined

Component Services Features

Other Resources

Introduction to COM and COM+

Automation

Automation is COM-based technology that enables dynamic binding to COM objects at run time. An Automation server is an application that allows its objects, methods, and properties to be controlled by other applications through Automation. An Automation client (also called an Automation controller) is an application that manipulates the objects, methods, and properties of another application (the Automation server) through Automation. An Automation object is an object that is exposed to other applications or programming tools through Automation interfaces.

A Transaction Integrator (TI) component is a type library that exposes an Automation interface. When you deploy a TI component in a COM+ application, you create an Automation server that can be used by any other COM-based application (Automation client). The Automation client application can call the methods on the Automation interface of a TI Automation server (for example, a TI component deployed in a COM+ application) to invoke a CICS or IMS transaction program (TP) on a mainframe host computer.

An Automation client application connects to a TI Automation server application in one of two ways, declarative binding or late binding:

- Declarative binding is similar to programming the speed dialer on your phone. You look up the number once, and program it into the phone. After you verify that it is valid, a single button dials the number.

- Late binding is similar to looking up the phone number in a phone book before each call, and then dialing the phone.

Both declarative binding and late binding produce the intended result. However, the speed dialer (declarative binding) is more efficient when you make a call, but looking up the number each time (late binding) ensures that the number you are calling is still correct.

In declarative binding, you declare a variable as an application-defined object type. A type library, object library, or dynamic-link library is then referenced in the Automation client Tools Reference. Declarative binding is another name for virtual function table binding or vtable binding. A vtable is a data structure that holds the addresses for the methods and properties of each object in an Automation server. The vtable is contained within the referenced library.

Late binding allows you to create an Automation object as an Object or Variant data type. Late binding uses the **IDispatch** interface. **IDispatch** has no preexisting information about the server that it is calling. It assumes during its compile phase that the code is correct. It then attempts at run time to execute the code and trap for run-time errors.

See Also
Concepts
COM Defined
Viewing COM Classes, Interfaces, and Methods
Other Resources
Introduction to COM and COM+

COM Objects

COM objects are entities in memory that can do specific tasks. Each COM object is an instance of a COM class. COM objects are not files, but can manage or represent files. (A COM object does not usually represent any particular file.) Their lifetimes are short. COM objects are deleted when program execution is terminated.

See Also

Concepts

COM Defined

Viewing COM Classes, Interfaces, and Methods

Other Resources

Introduction to COM and COM+

COM Methods

A COM method is a specific task that a COM object can perform. When a client requests that an object complete a specific task, that client calls a method of the object. For example, Winword.exe (a client) can call the **Spellchecker** method of a **Word.Document** (object) to run a spell-checking session. A Transaction Integrator (TI) Automation client application can call a method of an Automation server to run a mainframe transaction program (TP) and return the results.

See Also

Concepts

COM Defined

Viewing COM Classes, Interfaces, and Methods

Other Resources

Introduction to COM and COM+

COM Interfaces

A COM interface is a specific, immutable set of COM methods. COM clients cannot detect the internal workings of the objects they are using. Therefore, it is important for clients and objects to communicate about and agree on the functionality that an object supplies to a client. This agreement, frequently called a contract, is implemented in software by a COM interface.

To facilitate communication, the COM interface supplies a way for clients to query objects whether they implement specific sets of methods. For example, by way of the interface, COM allows a client to query an object whether it implements a set of file management methods, **FileSave** and **FileOpen**. An interface is named by an Interface ID (IID), which is a globally unique identifier (GUID), a 16-byte number. It is a COM rule that an object must either implement an interface or not implement it. There is no in-between state.

COM rules require that all COM objects must implement the **IUnknown** interface. **IUnknown** is used for the following:

- **Lifetime management.** To determine when objects are no longer in use and thus can be destroyed.
- **Versioning.** To determine which specific interfaces an object supports.

Objects that implement the **IDispatch** interface are scriptable, or Automation compatible; scripting clients such as Windows Script Host (WSH) can use them. Objects that do not implement **IDispatch** cannot be scripted; that is, scripts cannot call them. Objects that do not implement **IDispatch** cannot be Automation servers; client applications cannot use their services. In both cases, this inability is due to the clients need to know certain details about objects before they can call them.

Automation is COM-based technology that enables dynamic binding to COM objects at run time. When a client COM application is compiled, it does not have all the information about the services offered by the Automation servers that it will be using. Later, during run time, the client application uses an Automation server's **IDispatch** interface to find detailed information about the services that the Automation server offers.

After a TI component is associated with a specific remote environment (RE), and then dropped into a COM+ application, that application (or package) becomes an Automation server with an **IDispatch** interface. Then a client COM application can call upon the TI Automation server to operate (automate) a mainframe-based transaction program (TP) and return the results to the calling client COM application.

See Also
Concepts
COM Defined
Viewing COM Classes, Interfaces, and Methods
Other Resources
Introduction to COM and COM+

COM Classes

A COM class is a definition in source code that can be used as a template for creating objects that implement the **IUnknown** interface and expose only their interfaces to clients.

Different objects implement different methods. COM classes ensure that a given client connects to exactly the right kind of object. Because each COM object is an instance of a specific COM class, COM classes actually classify objects. For example, one class of object is the **Word.Document** class. Any object in this class can manage Microsoft Word .doc files.

Each COM class has two identifiers, the programmatic identifier (ProgID) and the class ID (CLSID). When a client tells COM that it needs use an object, the client always specifies a ProgID or CLSID so that COM knows which class of object to create.

A ProgID is a human-readable registry entry that can be associated with a CLSID. You can use the ProgID within programs to create an instance of a COM class. For example, to create an instance of a COM class that has a ProgID of **Component.MyComponent**, use the following Visual Basic statement:

```
Set Component = CreateObject("Component.MyComponent")
```

A ProgID must not contain more than 39 characters, start with a digit, nor contain any punctuation (including underscores) other than periods. The pieces are separated by periods; no spaces occur within the ProgID. For example, Word.Document.6 is a valid ProgID. The typical format of a ProgID is:

```
<Library>.<ComponentName>.<VersionNumber>
```

Although the ProgID is human-readable, it should be used only in programs, and not displayed in the user interface. If you need a short displayable string for an object, call **IOleObject::GetUserType**.

A ProgID identifies a class, but with less precision than does a CLSID. The registry entry appears as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\<ProgID> = <CLSID>
```

You can use a ProgID in programming situations where it is not possible to use a CLSID. The ProgID is not guaranteed to be unique, so use them only where name collisions are manageable.

The CLSID is a globally unique identifier (GUID), such as the following that uniquely identifies the COM class:

```
{71AFDE33-F81B-11d2-B12B-00C04F8C2F72}
```

You can generate a GUID with a program such as Uuidgen.exe.

See Also

Concepts

COM Defined

Viewing COM Classes, Interfaces, and Methods

Other Resources

Introduction to COM and COM+

COM Components and TI Components

A COM component is a binary file (.dll or .exe) containing code for creating a COM object and servicing method calls to it. The COM component can contain code for one or more Transaction Integrator (TI) components, class factories, COM classes, registry-entry mechanisms, type libraries, and so on.

COM-based programs and applications are distributed and deployed as COM components. Each COM component uses interfaces to interoperate with each other in much the same way that a person uses an application user interface to interoperate with an application. Each COM interface contains one or more methods.

A TI component is a type library that acts as a proxy for a mainframe transaction program (TP). You can easily change a TI component into a true COM component by deploying it in a COM+ application. TI components all share the same implementation DLLs but are distinguished by their type libraries. Therefore, when you deploy a TI component in a COM+ application, it is the specific type library (.tlb file) that is registered.

The basic unit of distribution for COM is a component. You can deploy COM components on a computer other than your own. This is what distinguishes COM components from COM objects, which reside in memory and thus cannot be copied from computer to computer.

Deploying a COM component is a two-step process. First, copy the component onto the target computer. This is a file copy operation. Then register the component, that is, configure COM to recognize the component. This configuration information is stored in the registry (and in certain cases in a database called regdb.) In TI, registration is automatic. As soon as you drop a TI component into a COM+ application, the resulting TI Automation server is automatically registered on that computer.

Because TI Manager includes Component Services to deploy or remove TI components, TI component deployment is a matter of dragging your TI component into a COM+ application.

See Also
Concepts
COM Defined
Other Resources
Introduction to COM and COM+

Viewing COM Classes, Interfaces, and Methods

Several tools are available for viewing the COM classes, interfaces, and methods that are deployed on your computer:

- **Component Services (COM+) in Windows 2000**

  Component Services (COM+) is a core part of Windows 2000. A class that has not been configured to use Component Services does not show up in Component Services.

- **OLE/COM Object Viewer (OLE View)**

  This tool shows all the deployed classes and gives comprehensive information about each class, with the exception of Component Services configuration information. It is available free at http://go.microsoft.com/fwlink/?LinkId=12793.

- **DCOMCNFG**

  This tool is present on any computer that has distributed COM (DCOM) installed, including all installations of Windows Server 2003, Windows XP Professional, and Windows 2000 Server. DCOMCNFG displays only a subset of the deployed classes, and shows much less information about each class than does OLEVIEW. Its main advantages are that it is automatically installed with DCOM and its graphical user interface (GUI) is less cluttered than the OLEVIEW GUI.

See Also
Concepts
COM Defined
COM Objects
COM Methods
COM Interfaces
COM Classes
Other Resources
Introduction to COM and COM+

Windows Script Host COM Client

Microsoft Windows contains utility COM objects that you can use to manage system configuration, users, printers, Active Directory directory service, and so on. Windows Script Host is a client that controls those Windows objects through scripts. Scripts give administrators much more administrative power than batch (.bat) files afford. The administrator writes a script, a file ending with .vbs or .jvs, with content much like a .bat file. The administrator then double-clicks the script to load Windows Script Host and run the script.

The following is an example of a short Microsoft Visual Basic Scripting Edition (VBScript) client script. This script has only three lines. The first line establishes a connection to the COM object. The second line calls a method, and the third line displays the result.

```
Set SaySvrObject = CreateObject("Saysvr.Sayclass")
Speech = SaySvrObject.SaySomething
MsgBox Speech
```

In this script:

- **Saysvr.Sayclass** specifies the COM class.
- **CreateObject** is the script's way of requesting that COM give it a reference to the object.
- **SaySvrObject** represents the reference to the object. After this line is completed, the script will have a live reference to the COM object.
- On the right side of the second line, the script uses its **SaySvrObject** reference to call the **SaySomething** method, which returns a string to the script.
- Speech represents the string returned by the script.
- The third line displays the Speech string in a Message Box.

See Also
Concepts
COM Defined
COM Components and TI Components
Viewing COM Classes, Interfaces, and Methods
Other Resources
Introduction to COM and COM+

Data Types

The Transaction Integrator (TI) run-time environment manages the process of converting host data types used by a host-based transaction program (TP) to and from the Automation data and data types used by a Microsoft Visual Basic or other Windows-based programming language.

The topics in this section give you tips for handling different types of data, and they explain how data types are converted and used in both IBM host and Automation programming languages.

TI supports Automation, COM aggregate, and mainframe data types. For a complete listing of the supported data types, see Supported TI Data Types

In This Section

Supported TI Data Types

Transaction Integrator (TI) supports the data types used by COM, .NET, COBOL, and Report Program Generator (RPG).

In This Section

Supported Automation Data Types

Automation data types are simple types that you can put into a Variant data type.

1-byte unsigned Integer (BYTE)
An Integer data type that has a positive value ranging from 0 through 255.

2-byte signed Integer
An Integer data type that can be either positive or negative. The most significant bit is the sign bit: 1 for negative values and 0 for positive values. The storage size of the integer is 2 bytes. A 2-byte signed Integer can have a range from –32,768 through 32,767.

4-byte signed Integer
An Integer data type that can be either positive or negative. The most significant bit is the sign bit: 1 for negative values and 0 for positive values. The storage size of a 4-byte signed Integer is 4 bytes. A 4-byte signed Integer can have a range from –2,147,483,648 through 2,147,483,647.

4-byte Real (Single)
A single-precision 32-bit (4-byte) floating-point Real data type (often called a Single). It ranges in value from –3.402823E38 through –1.401298E-45 for negative values, from 1.401298E-45 through 3.402823E38 for positive values, including 0.

8-byte Real (Double)
A double-precision 64-bit (8-byte) floating-point Real data type (often called a Double). It ranges in value from –1.79769313486232E308 through –4.94065645841247E-324 for negative values, from 4.94065645841247E-324 through 1.797693134862325E308 for positive values, including 0.

Boolean
An expression that can be evaluated either true (nonzero) or false (0). (You can use the keywords True and False to supply the values of –1 and 0, respectively.) The field data type Yes/No is Boolean and has the value of –1 for Yes and 0 for No. Several property page settings are Boolean, including Yes/No, True/False, and On/Off.

Variable-length String (BSTR)
A fundamental data type that holds character information. A String value can contain approximately 65,535 bytes (64 KB). It can be fixed-length or variable-length, and it contains one character per byte. Fixed-length Strings are declared to be a specific length. Variable-length Strings can be any length up to 64 KB, less a small amount of storage overhead. A variable length string can range from 0 to approximately 2 billion bytes (characters).

Currency
An 8-byte fixed-point data type that is useful for calculations involving money or for fixed-point calculations in which accuracy is extremely important. This data type is used to store numbers with up to 15 digits to the left of the decimal point and 4 digits to the right. Currency values can range from –922,337,203,685,477.5808 to 922,337,203,685,477.5807.

Date
A data type that is used to store a date, a time, or a date and time as a 64-bit Real number. The value to the left of the decimal point represents a date, and the value to the right of the decimal point represents a time. The Date data type values can range from January 1, 1000 to December 31, 9999.

Decimal
A data type that stores a signed, exact numeric value described as the number of digits appearing before and after a decimal point, with a maximum total of 18 digits.

See Also
Reference
Supported COM Aggregate Data Types
Supported .NET Aggregate Data Types
Supported COBOL Data Types
Supported RPG Data Types
Supported RPG Keywords
Other Resources
Supported TI Data Types

Supported COM Aggregate Data Types

Array

A set of sequentially indexed elements having the same intrinsic data type. Transaction Integrator (TI) supports an array of any of the Automation data types. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements. TI supports multidimensional arrays. However, only the outermost array of a multidimensional array can vary in size; all the other arrays must be fixed in size. For more information, see Arrays.

User-Defined Type (UDT)

A data type that is defined in a program. A UDT can contain any of the Automation data types as well as arrays of those data types, recordsets, and nested UDTs. A UDT generally contains many different data types that are defined by the programming language used. In COBOL, UDTs are called RECORDS as are any declarations that contain lower-level numbers. For more information, see User-Defined Types.

Recordset

An Automation object, also known as an ActiveX Data Object (ADO) recordset, is the equivalent of a fixed or unbounded table that contains simple types in COBOL or report program generator (RPG) data declarations. After you have a recordset object, you can call methods on that object to gain access to its rows. Recordsets supported by TI have disconnected client-side cursors. For more information, see Recordsets and Datatables.

See Also
Other Resources
Supported TI Data Types

Supported .NET Aggregate Data Types

Datatable

A .NET Framework object that is the equivalent of a fixed or unbounded table that contains simple types in COBOL or Report Program Generator (RPG) data declarations. After you have a **DataTable** object, you can call methods on that object to gain access to its rows. Datatables supported by TI have disconnected, client-side cursors. For more information, see Recordsets and Datatables.

Structures

A user-defined value type. Like a class, structures can contain constructors, constants, fields, methods, properties, indexers, operators, and nested types. Unlike classes, however, structures do not support inheritance.

The distributed program call (DPC) programming model for the AS/400 supports:

- Only single-level .NET structures.
- Arrays of .NET structures.

The DPC programming model for the AS/400 does not support:

- Nesting of structures.
- Arrays within structures.
- Variable sized structures in which the last parameter is a string.

Array

A set of sequentially indexed elements that have the same intrinsic data type. Transaction Integrator (TI) supports an array of any of the .NET data types in this topic. Each element of an array has a unique identifying index number. Changes made to one element of an array do not affect the other elements. TI supports multidimensional arrays. However, only the outermost array of a multidimensional array can vary in size; all the other arrays must be fixed in size. For more information, see Arrays.

See Also

Other Resources

Supported TI Data Types

Supported COBOL Data Types

COMP-1

A 4-byte, single precision, floating-point Real data type that specifies internal floating-point items. The sign is contained in the first bit of the leftmost byte, and the exponent is contained in the remaining seven bits of that byte. The remaining three bytes hold the mantissa.

COMP-2

An 8-byte, double precision, floating-point Real data type that specifies internal floating-point items. The sign is contained in the first bit of the leftmost byte, and the exponent is contained in the remaining seven bits of the first byte. The remaining seven bytes hold the mantissa.

COMP-3 Packed Decimal

A packed decimal data type that specifies internal decimal items stored in packed decimal format. In the packed decimal format, each byte in a field represents two numeric digits except for the rightmost byte. The rightmost byte holds one digit and the sign. In other words, there are two digits in each character position except for the trailing character position that is occupied by the low-order digit and sign. The item can contain any of the digits from 0 through 9, plus a sign, to represent a value not exceeding 18 decimal digits. For example, the decimal value +123 is represented in two bytes as 0001 0010 0011 1100 in packed decimal format. For more information see, Zoned Decimal or Packed Decimal Data Types.

DISPLAY Zoned Decimal

An unpacked decimal data type that specifies internal decimal items stored in zoned decimal format. Zoned decimal format is synonymous with unpacked decimal format, which is a format for representing numbers where each digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the least significant byte. Bits 0 through 3 of all bytes other than the least significant byte contain 1s (hex F). For example, the decimal value +123 is represented in three bytes as 1111 0001 1111 0010 1100 0011 in zoned decimal format. For more information see, Zoned Decimal or Packed Decimal Data Types.

DATE and TIME

Specifies a date and time by using group item of two PIC 9(7) COMP-3 Packed Decimal value.

TIME only

Specifies a time by using a PIC 9(7) COMP-3 Packed Decimal value.

DATE only

Specifies a date by using a PIC 9(7) COMP-3 Packed Decimal value.

PIC X

Specifies a single character in an Extended Binary Coded Decimal Interchange Code (EBCDIC) character string. EBCDIC is the native representation for character data on mainframes and AS/400s. Unicode is the native representation for character data on Windows-based platforms.

PIC X No Translation

Specifies a single COBOL character in an EBCDIC character string that is handled as if it were binary data. In other words, there is no translation from EBCDIC to Unicode or from Unicode to EBCDIC.

PIC G

Specifies a double-byte EBCDIC string.

PIC S9(4) COMP (Integer 16-bit)

Specifies an integer that is 16 bits, or 2 bytes, in length.

PIC S9(9) COMP (Integer 32-bit)

Specifies an integer that is 32 bits, or 4 bytes, in length.

See Also

Other Resources

Supported TI Data Types

Supported RPG Data Types

Transaction Integrator (TI) supports the data types in Report Program Generator (RPG) IV. The letter in parentheses is the internal representation of the data type.

- Character (A)
- Graphic (G)
- Numeric - Integer format (I)
- Numeric – Packed decimal format (P)
- Numeric - Zoned format (S)
- Numeric - Unsigned format (U)
- Float (F)
- Date (D)
- Time (T)
- Timestamp (Z)

See Also

Reference

Supported Automation Data Types
Supported COM Aggregate Data Types
Supported .NET Aggregate Data Types
Supported COBOL Data Types
Supported RPG Keywords

Other Resources

Supported TI Data Types

Supported RPG Keywords

Transaction Integrator (TI) supports the following Report Program Generator (RPG) keywords:

- CONST
- DATFMT
- DIM
- LIKE
- LIKEDS
- OCCURS
- PACKEVEN
- TIMFMT

TI does not support the following RPG keywords:

- ALIGN
- ALT
- ALTSEQ(*NONE)
- ASCEND
- BASED
- CTDATA
- DESCEND
- DTAARA
- EXPORT
- EXFLD
- EXTFMT
- EXTNAME
- EXTPGM
- EXTPROC
- FROMFILE
- IMPORT

- INZ
- NOOPT
- OPDESC
- OVERLAY
- PERRCD
- PREFIX
- PROCPTR
- STATIC
- TOFILE
- VALUE

See Also

Reference

Supported Automation Data Types
Supported COM Aggregate Data Types
Supported .NET Aggregate Data Types
Supported COBOL Data Types
Supported RPG Data Types

Other Resources

Supported TI Data Types

Data Type Conversion

The Transaction Integrator (TI) run-time environment automatically converts data types between host-based COBOL or Report Program Generator (RPG) data types and the COM-based Automation data types that a Windows-based programming language like Visual Basic uses. The automatic conversion is based on information that you define in TI Project when you design and create a TI component (type library). This information is stored with the TI component and used by the TI runtime to convert the parameters of a method from the representation understandable by a COM-based or .NET-based programming language into the representation understandable by a host transaction program (TP).

Use TI Project to associate each Automation data type with each COBOL or RPG data type used in the host TP. TI provides default mappings between standard Automation data types and COBOL or RPG data types. You can either accept the default mappings or override the default with other mappings supported by TI. TI Project stores the conversion map in the TI component type library (.tlb) file. This conversion map is used to:

- Handle data moving between the TP and the TI component.
- Convert a TI component into a TP (export the host definition).
- Convert a TP into a TI component (import the host definition).

If a parameter used in a method call is not strictly typed, the TI run-time environment attempts to coerce the data type it receives into the data type it expects. If that coercion is successful, the call proceeds. If it is unsuccessful, an error is returned.

At run time, when a client application uses the TI Automation interface to call a method of the TI Automation server, the TI run-time environment uses the conversion map to handle the actual data conversion of the in and in/out parameters being sent to the mainframe TP. After TI converts the Microsoft® Windows® data, TI reformats the method call as a host system APPC/LU 6.2 or TCP/IP message. Then TI uses Microsoft Host Integration Server 2009 SNA or TCP/IP connectivity to forward the message to the mainframe. When the mainframe TP returns the in/out and out parameters, TI reformats the message for the return to Windows, converts the host data into Windows data, and returns the return value and parameters to the client application.

The choice of language or code page you made in TI Manager when you defined the remote environment (RE) determines which code page is used to convert from UNICODE (on the Automation side) to Extended Binary Coded Decimal Interchange Code (EBCDIC) (on the mainframe side). When you create an RE in TI Manager, you can either select a language to accept the default code page for that language, or select a specific code page.

If you need to convert to different target code pages (if you have, for example, target mainframes in different countries or regions), you need to set up an RE for each target because TI does not support conversions requiring the use of locale.

You can use TI Project to import COBOL or RPG, or to manually enter method descriptions to create Automation methods. When you import COBOL or RPG, each supported COBOL data type has a default Automation type. When you manually create a method, each Automation data type has a default host data type associated with it.

If you do not want to use a default Automation data type, you can use TI Project to change the Automation data type manually. If the new Automation type is compatible with the existing host data type, the existing COBOL or RPG data type is left unchanged. If it is not compatible, the host data type is changed, thus affecting your mainframe program.

In This Section

- Zoned Decimal or Packed Decimal Data Types
- Converting Data Types from RPG to OS/400 COBOL

Converting Data Types from Automation to OS/390 COBOL

Use the tables in this topic as a guide to specify how you want TI to handle conversions from Automation data types to COBOL data types. For more information about the specific data types, see Supported TI Data Types.

Use the following code key to interpret the codes in the **Possible Conversion Errors** and **Required Property Settings** columns in each table.

Code	Description
r	Range errors possible.
b	Possible loss of precision due to base 2 to base 16 conversion.
p	Possible loss of precision due to picture format scale specification.
m	Mapping errors possible.
f	yyyyddd and hhmmss.
A	You must specify **Truncate**, **Round**, or **Error** under **Error handling** on the **COBOL Definition** tab of the property page.
C	You must specify the precision and scale by filling in the **Digits left** and **Digits right** boxes on the **COBOL Definition** tab of the property page.
D	You must specify the string width by filling in the **Size** box on the **COBOL Definition** tab of the property page.
E	Unicode or EBCDIC mapping information is required, such as a code page.
F	For arrays whose length is less than the maximum specified, you must specify **Size of Filler** under **Trailing filler** on the **COBOL Definition** tab of the property page.
G	You must specify how to deal with strings. Click **Space Padded** or **Null terminated** under **String Delimiting** on the **COBOL Definition** tab of the property page. Then click **Truncate** or **Error** under **Error handling** on the **COBOL Definition** tab of the property page to specify what TI should do if the string is too long.
H	Maximum size is required.
I	Localization is required.
J	Optional SO and SI insertion and deletion is supported.

The following table shows the defaults that TI uses for converting Automation data types to COBOL data types.

Default

From Automation data type	To OS/390 COBOL data type	Possible conversion errors	Required property settings
1-byte unsigned Integer	PIC X No Translation	None	None
2-byte signed Integer	PIC S9(4) COMP (Integer 16-bit)	None	None
4-byte signed Integer	PIC S9(9) COMP (Integer 32-bit)	None	None
4-byte Real (Single)	COMP-1	br	None
8-byte Real (Double)	COMP-2	br	None
Boolean	PIC S9(4) COMP (Integer 16-bit)	None	None
Variable-length String	PIC X	m	DEG
Currency	COMP-3 Packed Decimal	pr	C

Date (date and time)	COMP-3 Packed Decimal	pf	CI
Date (date only)	COMP-3 Packed Decimal	pf	CI
Date (time only)	COMP-3 Packed Decimal	pf	CI
Decimal	COMP-3 Packed Decimal	pr	C
Array (any data type)	OCCURS fixed TIMES	None	FH

Note

When you convert whole or fractional numbers from Visual Basic Single or Visual Basic Double data types to Packed Decimal or distributed program call (DPC) Zoned Decimal data types, TI is limited to a precision from 1 through 18 digits left of the de cimal point (for example, 1.2345678901234567E+17). When you convert fractional numbers Packed Decimal or DPC Zoned Decimal data types, you should convert to Visual Basic Decimal data type.

The following table shows the other supported data type mappings that you can set in TI Project to override the defaults presented in the previous table.

Supported in Transaction Integrator

From Automation data type	To OS/390 COBOL data type	Possible conversion errors	Required property settings
1-byte unsigned Integer	PIC S9(4) COMP (Integer 16-bit)	None	None
1-byte unsigned Integer	COMP-3 Packed Decimal	None	C
2-byte signed Integer	COMP-3 Packed Decimal	None	C
2-byte signed Integer	DISPLAY Zoned Decimal	None	C
4-byte signed Integer	COMP-3 Packed Decimal	None	C
4-byte signed Integer	DISPLAY Zoned Decimal	None	C
4-byte Real (Single)	PIC S9(4) COMP (Integer 16-bit)	p,r	None
4-byte Real (Single)	PIC S9(9) COMP (Integer 32-bit)	p,r	None
4-byte Real (Single)	COMP-3 Packed Decimal	p,r	C
4-byte Real (Single)	DISPLAY Zoned Decimal	p,r	C
8-byte Real (Double)	PIC S9(4) COMP (Integer 16-bit)	p,r	None
8-byte Real (Double)	PIC S9(9) COMP (Integer 32-bit)	p,r	
8-byte Real (Double)	COMP-3 Packed Decimal	p,r	C
8-byte Real (Double)	DISPLAY Zoned Decimal	p,r	C
Boolean	PIC S9(9) COMP (Integer 32-bit)	None	None
Boolean	COMP-3 Packed Decimal	None	C
Variable-length String	PIC G	m	DEGJ
Currency	PIC S9(?)V9(?) COMP (16-bit)	pr	None
Currency	PIC S9(?)V9(?) COMP (32-bit)	pr	None
Currency	DISPLAY Zoned Decimal	pr	C
Decimal	PIC S9(?)V9(?) COMP (16-bit)	pr	None

Decimal	PIC S9(?)V9(?) COMP (32-bit)	pr	None
Decimal	DISPLAY Zoned Decimal	pr	C
Array (any data type)	OCCURS DEPENDING ON	None	FH

Note

When you convert whole or fractional numbers from Visual Basic Single or Visual Basic Double data types to Packed Decimal or DPC Zoned Decimal data types, TI is limited to a precision of 1 to 18 digits left of the decimal point (for example, 1.2345678901234567E+17).

The following table shows additional supported data type mappings that the TI run-time environment supports.

Supported only by the TI run-time environment

From Automation data type	To OS/390 COBOL data type	Possible conversion errors	Required property settings
1-byte unsigned Integer	PIC S9(9) COMP (Integer 32-bit)	None	None
1-byte unsigned Integer	DISPLAY Zoned Decimal	None	C
Boolean	DISPLAY Zoned Decimal	None	C

No other data type conversions from Automation to COBOL are supported by TI at this time.

Note

When the COBOL usage is DISPLAY without a sign and you change the Automation type to String, the COBOL picture is changed to PIC X, which has the same internal data representation. The length remains the same and therefore does not affect your mainframe program.

See Also

Reference

Converting Data Types from OS/390 COBOL to Automation

Other Resources

Supported TI Data Types

Data Type Conversion

Converting Data Types from OS/390 COBOL to Automation

Use the tables in this topic as a guide when you set up the way you want Transaction Integrator (TI) to handle conversions from COBOL data types to Automation data types. For more information about the specific data types, see Supported TI Data Types.

Use the following code key to interpret the codes in the **Possible Conversion Errors** and **Required Property Settings** columns in each table.

Code	Description
r	Range errors possible.
b	Possible loss of precision due to base 2 to base 16 conversion.
p	Possible loss of precision due to picture format scale specification.
m	Mapping errors possible.
f	yyyyddd and hhmmss.
A	You must specify **Truncate**, **Round**, or **Error** under **Error handling** on the **COBOL Definition** tab of the property page.
C	You must specify the precision and scale by filling in the **Digits left** and **Digits right** boxes on the **COBOL Definition** tab of the property page.
D	You must specify the string width by filling in the **Size** box on the **COBOL Definition** tab of the property page.
E	Unicode or EBCDIC mapping information is required, such as a code page.
F	For arrays whose length is less than the maximum specified, you must specify **Size of Filler** under **Trailing filler** on the **COBOL Definition** tab of the property page.
G	You must specify how strings should be dealt with. Click **Space Padded** or **Null terminated** under **String Delimiting** on the **COBOL Definition** tab of the property page. Then click **Truncate** or **Error** under **Error handling** on the **COBOL Definition** tab of the property page to specify what TI should do if the string is too long.
H	Maximum size is required.
I	Localization is required.
J	Optional SO and SI insertion and deletion is supported.

The following table shows the defaults that TI uses when you import COBOL source code.

Default

From OS/390 COBOL data type	To Automation data type	Possible conversion errors	Required property settings
COMP-1	4-byte Real (Single)	b,r	None
COMP-2	8-byte Real (Double)	b,r	None
COMP-3 Packed Decimal	Currency	p	AC
COMP-3 Packed Decimal	Decimal	p	AC
DATE and TIME	Date	None	I
TIME only	Date	None	I
DATE only	Date	None	I
PIC X	Variable-length String	m	DEG

PIC X No Translation	1-byte unsigned Integer	None	None
PIC S9(4) COMP (Integer 16-bit)	2-byte signed Integer	None	None
PIC S9(4) COMP (Integer 16-bit)	Boolean	None	None
PIC S9(9) COMP (Integer 32-bit)	4-byte signed Integer	None	None
OCCURS fixed TIMES	Array	None	None

Note

When you convert fractional numbers from Packed Decimal or distributed program call (DPC) Zoned Decimal data types, yo u should convert to Visual Basic Decimal data type.

For COMP, COMP-3, and DISPLAY numeric COBOL data types, the default is based on the precision and scale shown in the following table. When COBOL uses DISPLAY without a sign and you change the Automation type to String, the COBOL picture is changed to PIC X, which has the same internal data representation. The length remains the same and therefore does not affect your mainframe program.

Precision and scale for OS/390 COBOL	To Automation data type
Precision 1-4, scale 0	2-byte signed Integer
Precision 5-9, scale 0	4-byte signed Integer
Precision 5-7, scale 3-7	4-byte Real
Precision 8-18, scale 3-18	8-byte Real
Precision 1-18, scale 1-2	Currency
Precision 10-18, scale 0	Decimal

The following table shows the other supported data type mappings that you can set in TI Project to override the defaults presented previously in this topic.

Supported in Transaction Integrator

From OS/390 COBOL data type	To Automation data type	Possible conversion errors	Required property settings
COMP-1	Array	None	None
COMP-2	Array	None	None
COMP-3 Packed Decimal	2-byte signed Integer	p,r	AC
COMP-3 Packed Decimal	4-byte signed Integer	p,r	AC
COMP-3 Packed Decimal	4-byte Real (Single)	p,r	AC
COMP-3 Packed Decimal	8-byte Real (Double)	p	C
COMP-3 Packed Decimal	Boolean	None	None
COMP-3 Packed Decimal	1-byte unsigned Integer	r	None
COMP-3 Packed Decimal	Array	None	None
DISPLAY Zoned Decimal	2-byte signed Integer	p,r	AC
DISPLAY Zoned Decimal	4-byte Real (Single)	p,r	AC
DISPLAY Zoned Decimal	8-byte Real (Double)	p,r	AC
DISPLAY Zoned Decimal	Currency	p,r	AC

DISPLAY Zoned Decimal	Decimal	p,r	AC
DATE and TIME	Array	None	None
TIME only	Array	None	None
DATE only	Array	None	None
PIC X	Array	None	None
PIC X No Translation	Array	None	None
PIC G	Variable-length String	m	DEGJ
PIC G	Array	None	None
PIC S9(4) COMP (Integer 16-bit)	1-byte unsigned Integer	r	None
PIC S9(4) COMP (Integer 16-bit)	Array	None	None
PIC S9(9) COMP (Integer 32-bit)	Boolean	None	None
PIC S9(9) COMP (Integer 32-bit)	1-byte unsigned Integer	r	None
PIC S9(9) COMP (Integer 32-bit)	Array	None	None
PIC S9(?)V9(?) COMP (16-bit)	4-byte Real (Single)	p,r	None
PIC S9(?)V9(?) COMP (16-bit)	8-byte Real (Double)	p,r	None
PIC S9(?)V9(?) COMP (16-bit)	Currency	p,r	None
PIC S9(?)V9(?) COMP (16-bit)	Decimal	p,r	None
PIC S9(?)V9(?) COMP (32-bit)	4-byte Real (Single)	p,r	None
PIC S9(?)V9(?) COMP (32-bit)	8-byte Real (Double)	p,r	None
PIC S9(?)V9(?) COMP (32-bit)	Currency	p,r	None
PIC S9(?)V9(?) COMP (32-bit)	Decimal	p,r	None
OCCURS DEPENDING ON	Array	None	None

Note

When you convert fractional numbers from Packed Decimal or DPC Zoned Decimal data types, you should convert to Visual Basic Decimal data type.

The following table shows additional supported data type mappings that the TI run-time environment supports.

Supported only by the TI run-time environment

From OS/390 COBOL data type	To Automation data type	Possible conversion errors	Required property settings
DISPLAY Zoned Decimal	1-byte unsigned Integer	None	AC
DISPLAY Zoned Decimal	4-byte signed Integer	None	AC
DISPLAY Zoned Decimal	Boolean	None	AC

No other data type conversions from COBOL to Automation are supported by TI at this time.

See Also
Reference
Converting Data Types from Automation to OS/390 COBOL
Other Resources
Supported TI Data Types

Data Type Conversion

Converting Data Types from Automation to RPG

Use the following table as a guide when you specify the way you want Transaction Integrator (TI) to handle conversions from Automation data types to Report Program Generator (RPG) data types.

TI Project default	RPG data type	Spec-ification	Field length	Field length meaning	Decimal places
Boolean (default)	Integer	I	5	digits	Blank
Boolean	Integer	I	10	digits	Blank
Boolean	Packed	P	3	digits	Blank,0
Byte (default)	Character	A	1	bytes	Blank
Byte	Unsigned	U	3-9	digits	Blank
Byte	Packed	P	3	digits	Blank,0
Byte	Integer	I	3-9	digits	Blank
Currency (default)	Packed	P	1-30	digits	Blank,0-4
Currency	Zoned	S	1-30	bytes	Blank,0-4
Currency	Binary	B	1-4	digits	Blank,0-4
Currency	Binary	B	5-9	digits	Blank,0-4
Date (Date)	*MDY	None	8	bytes	Blank
Date (Date)	*DMY	None	8	bytes	Blank
Date (Date)	*YMD	None	8	bytes	Blank
Date (Date)	*JUL	None	6	bytes	Blank
Date (Date)	*ISO	None	10	bytes	Blank
Date (Date)	*USA	None	10	bytes	Blank
Date (Date)	*EUR	None	10	bytes	Blank
Date (Date)	*JIS	None	10	bytes	Blank
Date (Time)	*HMS	None	8	bytes	Blank
Date (Time)	*ISO	None	8	bytes	Blank
Date (Time)	*USA	None	8	bytes	Blank
Date (Time)	*EUR	None	8	bytes	Blank
Date (Time)	*JIS	None	8	bytes	Blank
Date	Timestamp	Z	Number?	bytes	Blank
Decimal	Float	F	4	Bytes	Blank
Decimal	Float	F	8	Bytes	Blank
Decimal (default)	Packed	P	1-30	digits	Blank,0-30
Decimal	Zoned	S	1-30	bytes	Blank,0-30

Decimal	Binary	B	1-4	digits	Blank,0-4
Decimal	Binary	B	5-9	digits	Blank,0-9
Double (default)	Float	F	8	bytes	Blank
Double [1]	Packed	P	1-30	digits	Blank,0-30
Double [1]	Zoned	S	1-30	bytes	Blank,0-30
Double	Binary	B	1-4	digits	Blank,0-4
Double	Binary	B	5-9	digits	Blank,0-9
Integer (default)	Integer	I	1-5	digits	Blank
Integer	Packed	P	1-30	digits	Blank,0
Integer	Zoned	S	1-30	bytes	Blank,0
Integer	Binary	B	1-5	digits	Blank,0
Long (default)	Integer	I	1-9	digits	Blank
Long	Packed	P	1-30	digits	Blank,0
Long	Zoned	S	1-30	bytes	Blank,0
Long	Binary	B	1-9	digits	Blank,0
Single (default)	Float	F	4	bytes	Blank
Single [1]	Packed	P	1-30	digits	Blank,0-30
Single [1]	Zoned	S	1-30	bytes	Blank,0-30
Single	Binary	B	1-9	digits	Blank,0-9
String (default)	Character	A	1-32755	Bytes==Char	Blank
String	Graphic	G	1-16371	Char	Blank

Note

Note [1] in the preceding table indicates that when you convert whole or fractional numbers from Visual Basic Single or Visual Basic Double data types to Packed Decimal or distributed program call (DPC) Zoned Decimal data types, TI is limited to a precision from 1 through 18 digits to the left of the decimal point (for example, 1.2345678901234567E+17).

Note

While TI left-justifies all strings, the RPG MOVE command right-justifies all strings. If you are programming an RPG application, use the MOVEL or EVAL commands to perform the equivalent operation in RPG while manipulating a string. **See Also**

Supported TI Data Types

Converting Data Types from RPG to Automation

Data Type Conversion

Converting Data Types from RPG to Automation

Use the following tables as a guide when you set up the way you want Transaction Integrator (TI) to handle conversions from report program generator (RPG) data types to Automation data types. For more information about the specific data types, see Supported TI Data Types.

The following table describes the TI Project property abbreviations used in the data type tables that follow.

Abbreviation	Description
t	Truncate
e	Error
r	Round
sp	Space pad
nt	Null terminate
SO	Add leading shift in
SI	Add trailing shift out
PE	Pack even
TIP	TI Project

Pack even (PE) indicates that the definition specification uses the pack even option for RPG. PE indicates that the precision is an even number of digits when the From and To specification positions are used, which implies a byte count instead of a digit count and which might mean that the high-order digit position is ignored. For example, the following table shows how the number 256 in an RPG packed field is represented in internal memory.

Note
For purposes of this example, the number 256 fits in 2 bytes of memory in both the PE and No PE option.

Packed data type option	Byte 1	Byte 2		
	High-order byte high-order nibble	High-order byte low-order nibble	Low-order byte high-order nibble	Low-order byte Sign
No PE	2	5	6	0xf
PE	ignored	5	6	0xf

RPG data type	Spec-ification	RPG field length	TIP data type	TIP default error handling	TIP default field length	TIP default decimals	TIP default string handling
Character	A	1	Byte	None	None	None	None
Character	A	1-32755	String	t,e	80	None	sp,nt
Graphic	G	1-16371	String	t,e	80	None	sp
Binary	B	1-4	Currency	t,r,e	4	2	None
Binary	B	5-9	Currency	t,r,e	9	2	None
Binary	B	1-4	Decimal	t,r,e	4	2	None
Binary	B	5-9	Decimal	t,r,e	9	2	None
Binary	B	1-4	Double	t,r,e	4	2	None

Binary	B	5-9	Double	t,r,e	9	2	None
Binary	B	1-5	Integer	t,r,e	4	None	None
Binary	B	1-9	Long	t,r,e	9	None	None
Binary	B	1-9	Single	t,r,e	4	2	None
Integer	I	5	Boolean	None	None	None	None
Integer	I	10	Boolean	None	None	None	None
Integer	I	3-9	Byte	t,r,e	3	None	None
Integer	I	1-5	Integer	t,r,e	4	None	None
Integer	I	1-5	Long	t,r,e	9	None	None
Packed	P	3	Boolean	None	None	None	None
Packed	P	3	Byte	t,r,e,npe	3	None	None
Packed	P	1-30	Currency	t,r,e	8	2	None
Packed	P	1-30	Decimal	t,r,e	8	2	None
Packed	P	1-30	Double	t,r,e	8	2	None
Packed	P	1-30	Integer	t,r,e	3	None	None
Packed	P	1-30	Long	t,r,e	5	None	None
Packed	P	1-30	Single	t,r,e	8	2	None
Zoned	S	1-30	Currency	t,r,e	15	2	None
Zoned	S	1-30	Decimal	t,r,e	15	2	None
Zoned	S	1-30	Double	t,r,e	15	2	None
Zoned	S	1-30	Integer	t,r,e	5	None	None
Zoned	S	1-30	Long	t,r,e	9	None	None
Zoned	S	1-30	Single	t,r,e	15	2	None
Unsigned	U	3-9	Byte	t,r,e	3	None	None
Float	F	4	Decimal	t,r,e	None	None	None
Float	F	8	Decimal	t,r,e	None	None	None
Float	F	8	Double	t,r,e	8	None	None
Float	F	4	Single	t,r,e	4	None	None
Date	D	None	Date	None	None	None	None
Time	None	None	None	None	None	None	None
Time-stamp	None	None	None	None	None	None	None

RPG Date format name	Format	Range	Bytes
*MDY	mm/dd/yy	01/01/40 to 12/31/39	8

*DMY	dd/mm/yy	01/01/40 to 31/12/39	8
*YMD	yy/mm/dd	40/01/01 to 39/12/31	8
*JUL	yy/ddd	40/001 to 39/365	6
*ISO	yyyy-mm-dd	0001-01-01 to 9999-12-31	10
*USA	mm/dd/yyyy	01/01/0001 to 12/31/0000	10
*EUR	dd.mm.yyyy	01.01.0001 to 31.12.9999	10
*JIS	yyyy-mm-dd	0001-01-01 to 9999-12-31	10

RPG Time format name	Format	Range	Bytes
*HMS	hh:mm:ss	00:00:00 to 24:00:00	8
*ISO	hh.mm.ss	00:00:00 to 24:00:00	8
*USA	hh:mm AM or hh:mm PM	00:00 AM to 12:00 AM	8
*EUR	hh.mm.ss	00.00.00 to 24.00.00	8
*JIS	hh:mm:ss	00:00:00 to 24:00:00	8

RPG Timestamp Format	Bytes
yyyy-mm-dd-hh.mm.ss.mmmmmm	26

See Also
Reference
Converting Data Types from Automation to RPG
Other Resources
Supported TI Data Types
Data Type Conversion

Converting Data Types from COM to .NET

Transaction Integrator (TI) supports the following COM data types.

COM data type	.NET data type
Boolean	Boolean
Byte	Byte
Char	Char
Currency	<No corresponding type>
Date	DateTime
Decimal	Decimal
Double	Double
Integer	Int16
Long	Int32
Recordset	Datatable
Single	Single
String	String
User-defined type	Structure
Variant	Object

See Also

Reference

Converting Data Types from .NET to COM

Other Resources

Supported TI Data Types

Data Type Conversion

Converting Data Types from .NET to COM

Transaction Integrator (TI) supports the following .NET common language runtime (CLR) data types.

.NET data type	COM data type
Boolean	Boolean
Byte	Byte
Char	Char
Datatable	Recordset
DateTime	Date
Decimal	Decimal
Double	Double
Int16	Integer
Int32	Long
Object	Variant
Single	Single
String	String
Structure	User-defined type

See Also

Reference

Converting Data Types from COM to .NET

Other Resources

Supported TI Data Types

Data Type Conversion

Zoned Decimal or Packed Decimal Data Types

When it imports a host data declaration, Transaction Integrator (TI) converts Zoned Decimal (COBOL numeric PIC with DISPLAY or no USAGE, or RPG S data type) or Packed Decimal data types to Decimal or Currency Automation data types, respectively. Depending on the development application you are using, there might not be an equivalent for Decimal or Currency data types. If this is the case, use one of the following techniques to ensure that the data type works correctly with TI:

- Use language-supplied functions to manipulate the Automation types for Decimal or Currency.
- Within TI Project, if the data type has a fractional component, modify the method's parameter from the Decimal or Currency data type to the Floating Point Binary data type (double or single precision as appropriate). You can substitute a 16-bit or 32-bit binary Integer data type if the data declaration has no fractional component and the number of data declaration digits fits within the expected range.

Note
When you use the Floating Point Binary data type, the likelihood of a data conversion precision problem increases if fractions are involved. TI offers three options to handle data precision errors: Round (default), Truncate, or Error. The double-precision Floating Point Binary data type can handle host data declarations of up to fifteen digits.

See Also
Tasks
How to Use REDEFINES in COBOL
Concepts
COBOL FILLER

Converting Data Types from RPG to OS/400 COBOL

The Transaction Integrator (TI) run-time environment supports the use of OS/400 COBOL running on an AS/400 computer. The following table lists the Report Program Generator (to data type conversions supported by the TI run-time environment.

RPG data type	OS/400 COBOL data type
Type A	PIC X
Type G	PIC G
Type S	PIC S99v99 Display
Type L Note Type L is not supported by the TI run-time environment.	PIC S99v99 Display Sign Leading Separate
Type R Note Type R is not supported by the TI run-time environment.	PIC S99v99 Display Sign Trailing Separate
Type P	PIC S99v99 Comp
Type B Note Type B is not supported by the TI run-time environment.	PIC S99V99 Comp
Type I 5 digits	PIC S9(1) to S9(4) Comp-3
Type U 5 digits Note Type U 5 is not supported by the TI run-time environment. Use Type I 5 instead.	PIC 9(1) to 9(4) Comp-3
Type I 10	PIC S9(5) to S9(9) Comp-3
Type U 10 Note Type U 10 is not supported by the TI run-time environment. Use Type I 10 instea d.	PIC 9(5) to 9(9) Comp-3
Type F 4	COMP-1
Type F 8	COMP-2
Type D	None
Type T	None
Type Z	None

See Also

Other Resources

Supported TI Data Types

Data Type Conversion

Integer Data Type

Binary Integer data types apply to COBOL COMP and Report Program Generator (RPG) Integer data types. Automation data types support only a 16-bit or 32-bit Integer. Therefore, you should ensure that your client application is passing binary integer equivalents to the Transaction Integrator (TI) Automation server.

See Also

Other Resources

Data Types

Decimal Data Type in Visual Basic

If the client application uses Visual Basic 6.0 and includes the DECIMAL data type, the decimal value in Visual Basic must be defined as a Variant, and then assigned a decimal value by the Visual Basic function **CDec**. This must be done for the component to execute properly within the TI component.

The following is an example:

```
Dim xxx as Variant
xxx = CDec(27)
```

If the client application uses Visual Basic .NET, the DECIMAL data type is defined directly.

See Also
Other Resources
Data Types

Variant Data Type

The type of each element in a message is fixed and defined by the information in the component library. Because mainframe programs do not support the Variant data type, you must fix the type of each parameter at design time in Transaction Integrator (TI) Project. Microsoft Visual Basic Scripting Edition (VBScript), which is often used to create Active Server Pages (ASP) in Web-based applications, supports only the Variant data type. It does not accept declared variables. As a result, if your COM+ client application calls a TI Automation server and passes parameters with Variant data types, the TI run-time environment forces each Variant data type into the type for each parameter as defined in the TI component library.

The Variant data type is not supported in Visual Basic .NET. Visual Basic .NET supports defining data types as objects, and then casting the objects as data types. TI does not support variables defined as objects cast to data types. All method parameters must be defined initially as data types, not objects.

See Also

Other Resources

Data Types

Recordsets and Datatables

A recordset is an Automation object that is a fixed-size, bounded, or unbounded table that contains simple rows of host data declarations (data types). A datatable is a .NET object that is identical to a recordset in every regard, except that you cannot use the **NewRecordset** function with datatables. After you have a recordset or datatable object, you can call methods on that object to gain access to its rows.

A recordset or datatable is implemented on top of row sets by Remote Data Service (RDS), which is a part of Microsoft Data Access Components (MDAC) version 2.5. You can use the **RDSServer.DataFactory** object to create a recordset or datatable and use ActiveX® Data Objects (ADO) to update or read the recordset.

A recordset or datatable provides a means of presenting and manipulating tabular data. Currently, recordsets cannot be nested, cannot contain arrays, and cannot contain user-defined types (UDTs).

Support for recordsets and databales enables TI to support what is effectively an array of a structure (or a record, in COBOL terminology) as well as a structure. A structure is represented as a fixed-size recordset or datatable where each column in the row contains a single data element. To deal with mainframe programming issues, TI classifies recordsets and datatables as fixed-size, bounded, or unbounded, in reference to the number of rows contained in the recordset or datatable.

Important
The OS/400 distributed program calls (DPC) programming model supports only fixed size recordsets and datatables. The pro gramming model does not support unbounded recordsets and datatables, nor does it support the use of the OCCURS DEPE NDING ON clause, or variably-sized recordsets and datatables.

For fixed-size, bounded, and unbounded TI recordsets and datatables, the layout of all rows in a particular recordset is the same and is defined at design time by using TI Project. If a recordset or datatable is an output or return value from the mainframe, TI run-time environment uses the **RDSServer.DataFactory** object to create a recordset or datatable and ADO to fill the recordset or datatable with the rows of data returned from the mainframe program.

Such a recordset is a disconnected recordset with a cursor type of adOpenForwardOnly. To scan the recordset requires calling **MoveFirst** and **MoveNext** to move through the rows. The recordset can be updated in place, but because it is disconnected from the true data source (the data source manipulated by the mainframe program that returned the data), the updates are not propagated to the original data source.

NewRecordset is a function that is supplied automatically for all TI components. This function is called to create a disconnected recordset object that can be passed into a TI method call. **NewRecordset** is provided as a convenience for TI client applications; it is not required to pass a recordset to a TI component's methods. The function can be called only for input or input/output recordset objects. The TI run-time environment creates a recordset object when the parameter is an output recordset object.

In This Section

Example Using the New Recordset Call

The following example code uses the **NewRecordset** property to create a new disconnected recordset object, and it passes the object as an input parameter to a Transaction Integrator (TI) method by way of the **IDispatch** interface. The *bstrName* parameter in the **IDispatch** call is the name of the recordset object (for example, **Recordset1**) as specified in TI Project. The call returns the dispatch pointer to an ActiveX® Data Objects (ADO) database recordset object if the call is successful. If the call is not successful, a NULL pointer is returned.

```
IDispatch * NewRecordset(
  BSTR bstrName);

Dim pMyRecord as Object
Dim pMyComponent as Object
Dim varFields(2) as Variant
Dim varValues(2) as Variant
On Error Goto SomeErrorHandler

' Create an instance of the TI object
set pMyComponent = CreateObject("MyComponent.Interface1")

' Create an instance of a recordset object
set pMyRecord = pMyComponent.NewRecordset("Recordset1")

' Fill in some recordset fields
varFields(0) = CStr("member1")
varFields(1) = CStr("member2")
varValues(0) = CInt(99)
varValues(1) = CLng(999)

' Add the recordset fields and values to the recordset object
pMyRecord.AddNew varFields, varValues

' Call the method with the recordset object as a parameter
pMyComponent.Method1(pMyRecord)
```

The TI run-time environment performs some compatibility checks between the structure definition for the recordset in the component library and the recordset object seen at run time. Any incompatibility will result in an Automation exception.

Note

Because a recordset cannot be nested, cannot contain arrays, and cannot contain user-defined types (UDTs), a COBOL data st ructure that is imported into TI Project might be flattened, resulting in an increase in the number of parameters that must be handled on the Automation side.

See Also
Tasks
Recordset Creation When Importing COBOL
Concepts
Fixed-Size Recordsets
Bounded Recordsets
Unbounded Recordsets
RDS Recordset Requirements for Web Clients

Fixed-Size Recordsets

A fixed-size recordset holds a fixed number of rows that is exactly equal to the maximum number of rows that you specified at design time in Transaction Integrator (TI) Project. A parameter or return value can be defined as a fixed-size recordset for any of the TI programming models.

Among the three possible types of recordsets or datatables (fixed-size, bounded, or unbounded), the TI default is fixed-size.

If the COBOL data declaration contains an OCCURS DEPENDING ON clause, each OCCURS DEPENDING ON clause requires two parameters:

- The size of the OCCURS DEPENDING ON clause.
- The data representation (a single element or a COBOL record) within the OCCURS DEPENDING ON clause.

If the COBOL contains an OCCURS DEPENDING ON clause, when the recordset or datatable is returned to TI and processed at run time, the TI run-time environment looks for the value of the size parameter corresponding to the named parameter (from the OCCURS DEPENDING ON clause). When it sends a fixed-size recordset as an input parameter to the mainframe, the TI run-time environment sets the COBOL size parameter with the actual number of rows in the recordset.

The DPC programming model on the AS/400 platform supports a modified version of OCCURS DEPENDING. By using the OCCURS DEPENDING property to select a parameter to specify size for the array, DPC-invoked programs can receive and send arrays that are partially populated. For example, you might have a structure that is defined to occur 10 times and that you want to return to the client. However, there might be fewer than 10 occurrences that actually contain valid data (for example, if the array was filled from a database table that contained fewer than 10 rows). If you define the array using the OCCURS DEPENDING property, you do not have to fill the remaining occurrences with dummy data. The TI run-time environment will pass only the filled occurrences back to the client.

See Also

Tasks

Example Using the New Recordset Call
Using the OCCURS DEPENDING Clause to Define Variable-length Table
Using Variably Sized Rows
Using Bounded Final Fields
Recordset Creation When Importing COBOL

Concepts

Bounded Recordsets
Unbounded Recordsets
RDS Recordset Requirements for Web Clients

Other Resources

Data Transfer Options

Bounded Recordsets

A recordset that is the last input parameter or the last output parameter in a method can be bounded. This means its actual size (number of rows) can vary from zero up to the maximum number of rows specified at design time. A parameter or return value can be defined as a bounded recordset for all of the Transaction Integrator (TI) programming models except OS/400 distributed program calls (DPCs). The DPC programming model does not support bounded recordsets, datatables, or arrays.

A bounded recordset, datatable, or array must be the last parameter in a particular direction. The last input parameter can be bounded if it is a recordset or an array and there are no following input/output parameters. The last output parameter can be bounded if it is a recordset or an array and there are no following input/output parameters. An input/output parameter can be bounded if it is a recordset or an array and is the final parameter. The mainframe transaction program (TP) is responsible for sending a table of the correct size, depending on the number of elements or rows in use.

Whereas a fixed-size recordset must send the exact number of rows defined as the maximum within the recordset, the number of rows in a bounded recordset can vary from zero to the maximum.

Among the three possible types of recordsets (fixed-size, bounded, or unbounded), the TI default is fixed-size. To change a fixed-size recordset into a bounded recordset within Designer, set the property "variable sized final field" on the method that contains the parameter defined as a recordset to be bounded, to true.

See Also

Tasks

Example Using the New Recordset Call

Recordset Creation When Importing COBOL

Concepts

Fixed-Size Recordsets

Unbounded Recordsets

RDS Recordset Requirements for Web Clients

Unbounded Recordsets

The number of rows in an unbounded recordset or datatable is not specified (defined) before run time. A parameter or return value can be defined as an unbounded recordset or datatable for any of the following Transaction Integrator (TI) programming models:

- CICS User Data LU 6.2
- TCP Transaction Request Message User Data
- IMS User Data LU 6.2
- IMS Implicit
- IMS Explicit
- IMS Connect

Note
In the IMS Connect programming model, TI supports only output parameters as unbounded recordsets or datatable. In other words, you cannot send unbounded recordsets or datatables to the host from TI if you are using the IMS Connect model.

An unbounded recordset or datatable is always transmitted to or from the mainframe after all other data. TI supports at most one unbounded input parameter and one unbounded output parameter, and each must be the last parameter in a direction.

To make a fixed size recordset or database unbounded, use the designer to set the **Unbounded** property of the recordset to true. The property **Variable Sized Final Field** on the method associated with the unbounded recordset must be false.

After defining a recordset or datatable as unbounded, the designer **recordset** property class will contain a property called **variably sized rows**. Use this property to determine how the length of the row will be represented.

Unbounded recordsets or datatable that contain variably sized rows must receive the size of the recordset or datatable first, and then receive the data. Therefore, the size of the recordset or datatable must be sent first followed by a second send that contains the data. If TI expects to receive an unbounded recordset or datatable with variably sized rows, CICS must first send the length of the row, and then send the data for that row. These two steps are repeated for each row of the recordset or datatable. TI detects that there is no more data because CICS stops sending and an unbounded recordset or datatable must always be the last parameter sent to TI.

See Also

Tasks

Example Using the New Recordset Call

Recordset Creation When Importing COBOL

Concepts

Fixed-Size Recordsets

Bounded Recordsets

RDS Recordset Requirements for Web Clients

RDS Recordset Requirements for Web Clients

If you have a Web-based Automation client application that uses a Remote Data Service (RDS) recordset control (most likely bound to a grid control) to display a table returned from a mainframe transaction program (TP), RDS requires that:

- The returned table be described as a recordset when you import your host data declaration into Transaction Integrator (TI) Project.

- The recordset be the return value from the method. The RDS control uses the HTTP protocol to marshal the recordset from the TI component on the server to the Web browser.

When you import a host data declaration, you must describe the table as a recordset. For more information, see Recordset Creation When Importing COBOL.

When you create the Automation method for your TI component in TI Project, be sure to have the method return the recordset as a return type instead of defining the recordset as a parameter. If you are importing the host code, when TI Project requests that you select the return value item, select the group item level for your table in the host declaration. To manually create a method, first manually describe or import the recordset in TI Project. Then use the **Insert** menu item to add the method that the Web client will call. Select the recordset you just described from the data type submenu list.

There are other properties that you can set on the recordset that is being returned. For details, see the shortcut Help for the **Recordsets** property page.

See Also

Tasks

Example Using the New Recordset Call

Recordset Creation When Importing COBOL

Concepts

Fixed-Size Recordsets

Bounded Recordsets

Unbounded Recordsets

Recordset Creation When Importing COBOL

When you import COBOL in Transaction Integrator (TI) Project, you can choose between two ways to create recordsets and datatables: automatically and explicitly.

When you import COBOL to create or update an Automation method, TI Project automatically creates recordsets and datatables when it encounters a fixed or variable-length table (OCCURS clause). The value for the OCCURS clause's length specifier can be any value greater than or equal to 1. The table must conform to recordset rules, which means that it cannot contain arrays, and it cannot contain nested tables that cannot be flattened.

When you import COBOL to explicitly create or update a recordset or datatable, you can select any group item that conforms to recordset rules. For importing recordsets and datatables, the group item does not need to have an OCCURS clause. If an OCCURS clause is present, it is ignored.

The following COBOL source code was imported using the Import COBOLWizard. The INVOICES table was selected as the return value for the method.

```
01 CUSTOMER-DATA.
    05 CUSTOMER-NUMBER                PIC 9(9).
    05 INVOICES OCCURS 50 TIMES.
       10 INVOICE-NUMBER              PIC 9(9).
       10 INVOICE-DATE                PIC 9(7) COMP-3.
       10 INVOICE-AMOUNT              PIC S9(13)V9(2) COMP-3.
       10 INVOICE-DESCIPTION          PIC X(40).
    05 LAST-NAME                      PIC X(20).
    05 FIRST-NAME                     PIC X(20).
```

The resulting method was imported as:

```
GetInvoices(lCustomerNumber As Long, strLastName As String, strFirstName As String) As Obje
ct
```

The following Visual Basic code demonstrates how to call this method:

```
Dim objCustomer As Object   'Uses late binding
Dim objInvoices As ADODB.Recordset
Dim lCustomerNumber As Long
Dim iRow As Integer
Dim iCol As Integer
Dim strLastName As String
Dim strFirstName As String

'create an instance of the invoicing object
On Error GoTo ErrorHandler1
Set objCustomer = CreateObject("Customer.Invoicing.1")

lCustomerNumber = CLng(txtCustomerNumber)

'invoke the GetInvoices method
On Error GoTo ErrorHandler2
Set objInvoices = objCustomer.GetInvoices(lCustomerNumber _
    , strLastName, strFirstName)
'
' Transfer the Recordset data to a variant array in a single operation.
' This is efficient, but may not be suitable for larger Recordsets.
'
Dim Data As Variant

Data = objInvoices.GetRows
grdInvoices.Rows = UBound(Data, 2) + 1
grdInvoices.Cols = UBound(Data, 1) + 1
For iRow = 0 To UBound(Data, 2)
```

```
    grdInvoices.Row = iRow
    For iCol = 0 To UBound(Data, 1)
        grdInvoices.Col = iCol
        grdInvoices.Text = Data(iCol, iRow)
    Next iCol
Next iRow
```

See Also

Tasks

Example Using the New Recordset Call

Concepts

Fixed-Size Recordsets

Bounded Recordsets

Unbounded Recordsets

RDS Recordset Requirements for Web Clients

Arrays

Transaction Integrator (TI) supports multidimensional arrays for all programming models except the OS/400 distributed program calls (DPCs). Only the outermost array of a multidimensional array can vary in size. All other arrays must be fixed-sized. For the OS/400 DPC programming model, TI supports only single-dimension arrays.

Arrays in Automation are actually SAFEARRAYs that contain information about their bounds as well as the data for the array elements. SAFEARRAYs are mapped to fixed-size arrays on the mainframe. SAFEARRAYs have a variable size and require custom information to be marshaled to and from fixed-size arrays on the mainframe.

Arrays are created on the mainframe during the import process when a simple data type has one or more OCCURS clauses or DIM keywords. The OCCURS clause can represent a fixed or variable-length table. Although it is possible in COBOL to have nested OCCURS DEPENDING clauses, only the OCCURS DEPENDING length specifier for the outermost table dimension is supported by TI. TI Project ignores nested length specifiers.

The mainframe developer might want to reflect the fixed nature of the array through to the COM object. The error-handling features of the TI run-time environment allow the developer to specify that an exception be raised if an array is not of the expected size. However, the mainframe developer might also want to take advantage of the flexible nature of SAFEARRAYs. TI supports an additional parameter that specifies the actual run-time size of the array. This parameter is supported with the same behavior as the OCCURS DEPENDING ON clause in COBOL. The maximum size of the array remains fixed, but the developer can determine the actual number of elements in the array received from the client.

The following custom information can be associated with either single or multidimensional arrays:

- Maximum size (on each dimension) (Integer: default 10). This value is automatically set to "x" if the COBOL imported into TI Project contains an OCCURS DEPENDING x TIMES clause.

- If the COBOL imported into TI Project contains an OCCURS DEPENDING ON clause, the actual size of the array is associated with the parameter specified by "name".

See Also
Tasks
Visual Basic and Arrays

Visual Basic and Arrays

When you develop client applications with Microsoft Visual Basic Scripting Edition (VBScript), you cannot define methods that return arrays. If it is necessary to pass back array information, define an output parameter.

Transaction Integrator (TI) deals with multidimensional arrays in row major order. The outermost dimension of an array in COBOL equates to the leftmost dimension in an array on the Automation side. In Visual Basic, the convention is to use column major order for multidimensional arrays. Therefore, when you access multidimensional arrays in Visual Basic, remember that the array will be row major for TI purposes. This can be a problem if you use Visual Basic functions that deal with multidimensional arrays and expect those arrays to be in column major order. When you access individual array elements, however, this should not be a problem.

Note

Manually re-dimensioning a multidimension array that was generated by TI yields might cause the component to function in correctly.

Note

The Visual Basic command Option Base has no effect on COM SAFEARRAYs, such as those returned as Out or In/Out arrays f rom TI methods. These SAFEARRAYs always have a base value of 0, regardless of the option base setting.

The following COBOL sample shows a variable-length table:

```
01 UPDATEARRAY-INPUT-AREA.
   05 CUSTOMER-NUMBER                PIC 9(9) DISPLAY.
   05 LAST-NAME                      PIC X(20).
   05 FIRST-NAME                     PIC X(20).
   05 TABLE-LENGTH                   PIC 9(7) COMP-3.
   05 TABLE OCCURS 10 TIMES
         DEPENDING ON TABLE-LENGTH
         OF UPDATEARRAY-INPUT-AREA
                                    PIC S9(4) COMP.
```

TI Project creates the following method when you import the COBOL sample:

```
UpdateArray(lCustomerNo As Long, strLastName As String, strFirstName As String _
     , lcElements As Long, rgTable() As Integer)
```

The following Visual Basic code shows how to send and receive the table:

```
    Dim objCustomer As Object
    Dim i
    Dim rgTable() As Integer
    Dim lCustomerNo As Long
    Dim strLastName As String
    Dim strFirstName As String
    Dim lcElements As Long

    'create an instance of the invoicing object
    On Error GoTo ErrorHandler1
    Set objCustomer = CreateObject("Customer.Invoicing.1")

    lCustomerNo = 100231001
    strLastName = "Doe"
    strFirstName = "John"

    'Send over 5 elements
    lcElements = 5
    ReDim rgTable(lcElements)
```

```
For i = 0 To lcElements - 1
    rgTable(0) = i
Next i

'invoke the UpdateArray method
On Error GoTo ErrorHandler2
objCustomer.UpdateArray lCustomerNo, strLastName, strFirstName _
    , lcElements, rgTable

'lcElements now contains the number of elements returned in rgTable
```

See Also

Other Resources

Arrays

User-Defined Types

In general, a user-defined type (UDT) is equivalent to a structure in C, C++, or Report Program Generator (RPG) records or group item levels in COBOL, and UDTs in Visual Basic. UDTs can contain primitive elements, arrays of elements, recordsets, and nested UDTs.

You can use UDTs to define parameters. A parameter can be defined as a single UDT or as an array of UDTs. Likewise, UDTs can describe a method's return value. A method's return value can be defined as a single UDT or as an array of UDTs. However, you cannot use a UDT to describe a column within a recordset.

TI requires that if you use Visual Basic to build your UDTs, you must use Visual Basic version 6.0 or later.

Before you compile a Visual Basic application that calls a TI component that defines a UDT and that references that UDT within the Visual Basic application, you need to deploy the TI component to register it. If you compile the application before you deploy the TI component or if you fail to deploy the component at all, the Visual Basic client application will not start.

The distributed program call (DPC) programming model for the AS/400 supports:

- Only single-level UDTs and .NET structures.
- Arrays of UDTs and structures.

The DPC programming model for the AS/400 does not support:

- Nesting of UDTs and structures.
- Arrays within UDTs and structures.
- Variable sized UDTs in which the last parameter is a string.

In This Section

Requirements for UDT Support in Windows 2000

When Windows 2000 is installed, Transaction Integrator (TI) offers full functionality. Otherwise, TI restricts the use of user-defined types (UDTs). The functionality offered in this restricted mode is as follows:

- The TI Component Registrar recognizes that it is executing on a computer that does not have Service Pack 4 (SP4) installed. If the registrar is instructed to deploy a component when UDT support is not available, the registrar will search the type library to ensure that there are no UDTs. If UDTs are detected, the component deployment will fail and an error describing this failure will be written to the event log.

- TI Project recognizes that it is executing on a computer that does not have Windows 2000 installed. Although TI Project does not prevent UDTs within component descriptions, it warns users when UDTs are being used on a computer that does not have Windows 2000. Specifically, in TI Project, when the user attempts to save the type library, a UDT warning appears if UDTs are used in the component's description. This warning appears only after the first attempt to save a component description.

See Also
Tasks
Arrays Defined in UDTs with Visual Basic
Concepts
Using TI Project to Create UDTs
Creating UDTs by Importing Host Definitions

Using TI Project to Create UDTs

In Microsoft® Visual Studio® Transaction Integrator (TI) Project, you can create a user-defined type (UDT) manually by right-clicking the **User-Defined Type** folder, and then click **Add User-defined Type**. This automatically adds one member field of the type Integer. If only one member field exists in a UDT, you cannot delete that member field; you can only delete the entire UDT. You cannot delete a UDT if it is being used as a type by some other variable in the interface. The name of a UDT cannot be the same as a method name, recordset name, or any reserved names.

To add a member field to a UDT, right-click a previously defined UDT, and then click **Add User-defined Type Member**. A default member is created and can be modified by updating its properties in the **Properties** pane.

In addition to the **Automation** and **COBOL Definition** property pages for UDT member fields, an **Arrays** property page supports defining arrays of member fields, and a **Recordsets** property page supports definition of a maximum row count and options for recordset columns. The rules for arrays of recordsets are the same as for parameters. An OCCURS DEPENDING index must be one of the member fields above the array itself.

In TI Project, you can copy and paste UDT members from and to recordset members.

See Also

Tasks

Arrays Defined in UDTs with Visual Basic

Concepts

Requirements for UDT Support in Windows 2000

Creating UDTs by Importing Host Definitions

Creating UDTs by Importing Host Definitions

The parser phase of the host import function identifies any group-level clause as a possible user-defined type (UDT). UDTs are created when you specify creating or updating a UDT or as the result of creating a method that contains a UDT. When you create or update a method, there is an additional wizard page following the **Select Return Value** page for specifying that groups of data items are UDTs or recordsets. This page also enables you to cancel the selection of groups of data items as UDTs or recordsets. For backward compatibility, the default data type of a group-level clause is a recordset if the group level has an OCCURS keyword. Also for backward compatibility (that is, updating a method), if there is no OCCURS keyword, the group level is flattened by default.

See Also

Tasks

Arrays Defined in UDTs with Visual Basic

Concepts

Requirements for UDT Support in Windows 2000

Using TI Project to Create UDTs

Arrays Defined in UDTs with Visual Basic

In Visual Basic, if an array is defined within a user-defined type (UDT), you must re-dimension the array within the UDT before you populate the array. A UDT is a group of data types assigned to a single parameter, which is defined with an alias. In the following example, the UDT is defined within a Transaction Integrator (TI) type library, and not within the Visual Basic client.

The following example presents Visual Basic code that defines an array within a UDT:

```
*** Assign the udt defined within the comti typelib to a variable within the Visual Basic a
pplication.
Dim ClientUDT as UDT1
*** assign values to string data type within UDT1
ClientUDT.string = "ABCDEFG"
*** assign size before populating
ReDim ClientUDT.array1(10).
** start assigning values to array within UDT.
ClientUDT.array1(1) = 100
```

Note

In an array within a UDT, do not define the array within the Visual Basic client before you perform a re-dimension against the UDT. For all other data types within the UDT, you can immediately populate the values without having to define them within t he Visual Basic client data declarations.

Note

If you have an output or an in/out array inside a UDT, use a lower bound of zero for all array dimensions. Use zero because t ype libraries contain information only about array dimensions, not upper and lower bounds. Therefore, Convert has to choos e them arbitrarily when it creates the arrays.

Note

Report Program Generator (RPG) has additional limitations on the use of UDTs. RPG does not support:

- Nesting more than one level of arrays of UDTs.
- UDTs that contain recordsets within UDTs.

See Also
Concepts
Requirements for UDT Support in Windows 2000
Using TI Project to Create UDTs
Creating UDTs by Importing Host Definitions

Transaction Request Messages

When you use TCP/IP to communicate with CICS, the client sends the host a transaction request message (TRM) request containing the Transaction Program ID, User ID, Password, and other administrative data to be used by the host. CICS sends the client a TRM reply containing additional administrative data. The data in the TRM is independent from the actual program data to be exchanged with the Transaction Program (TP) on the host.

You can find templates for various TRMs at \installation directory\Microsoft Host Integration Server\system\TIM\MicrosoftTRMDefs.tim. Use Microsoft Visual Studio to open the file, and then expand the **User-Defined Types** node. The following TRMs are defined as UDTs:

- TRMIN_MSLink
- TRMOUT_MSLink
- TRMIN_MSCCS
- TRMIN_IBMCCS
- TRMOUT_CCS

You can also find templates for various enhanced listener messages (ELMs) at \installation directory\Microsoft Host Integration Server\system\TIM\MicrosoftELMDefs.tim. Use Visual Studio to open the file, and then expand the **User-Defined Types** node. The following ELMs are defined as UDTs:

- ELMIN_MSLink
- ELMOUT_MSLink
- ELMIN_MSCCS
- ELMIN_IBMCCS
- ELMOUT_CCS

You can create a TRM or ELM template in COBOL to assist with your programming by exporting the TRM or ELM definition.

To create a TRM template in COBOL

1. Open Visual Studio.
2. On the **File** menu, point to **Open**, and then click **File**.
3. In the **Open File** dialog box, navigate to <drive>:\Program Files\Microsoft Host Integration Server\System\TIM\, and then click either **MicrosoftTRMDefs.tim** or **MicrosoftELMDefs.tim**.
4. On the **File** menu, click **Export Host Definition**.
5. In the **Export Host Definition** dialog box, type or select the file name, and then click **Save**.

You can substitute a custom TRM (or ELM) for the default TRM (or ELM) created by the TI runtime. Use the COMTIContext parameter to pass custom context data.

See Also

Tasks

How to Pass a Custom TRM

Concepts

Using Custom TRMs and ELMs with COMTIContext

Using Custom TRMs and ELMs with COMTIContext

Transaction Integrator (TI) developers can pass custom formatted transaction request messages (TRMs) or enhanced listener messages (ELMs) from a client program to the CICS system and receive custom formatted TRMs or ELMs.

Custom TRMs or ELMs are passed through context data. The context data is contained in the optional *COMTIContext* parameter defined in the client application code and must be the final parameter in the method call. A TRM destined for the host must be defined as a user-defined type (UDT) for the COM model, or a structure for the .NET Framework model. The name of the UDT must begin with the characters TRMIN. A TRM reply from the host must also be defined as a UDT. The name of the UDT must begin with the characters TRMOUT. Examples of valid TRM names are: TRMINMyVeryOwn, TRMINStandard, TRMOUTMyVeryOwn, and TRMOUTStandard.

The type library or structure can contain multiple TRM definitions, but you should include only one TRM for each direction (that is, one TRMIN and one TRMOUT) in the COMTIContext parameter in a single method call. For example, in Visual Basic each COMTIContext array is declared as a single dimension dynamic array of variants (that is, the number of occurrences is not specified).

If you define multiple TRMs for the same direction, the TI run time uses only the first TRM it encounters in the context array. (In some circumstances, the first TRM encountered might not always be the first one that you added to the context array). The TI run time ignores the remaining TRMs in the array until the TRM in use is destroyed. To ensure that the TI run time uses the correct TRM, do not add multiple TRMs intended for the same direction to a context array.

Note

The client application that manipulates the Context array must be able to access the appropriate file at run time. If you are using Visual Basic6.0, the application must be able to access COMTIContext.dll. If you are using Visual Basic .NET, the application must able to access Microsoft.HostIntegration.TI.ClientContext.dll.

Note

When you use Visual Basic .NET, the data structure used as a custom TRM must be associated to a parameter within the assembly. Therefore, you must create a dummy method within the assembly, a parameter assigned to the method, and the data structure to be used as the TRM. Failure to do so prevents you from referencing the structure within the Visual Basic .NET application. Associating a UDT to a method was not required in Visual Basic 6.0 because Visual Basic 6.0 allowed referencing of UDTs not associated with methods.

See Also

Tasks

Transaction Request Messages

How to Pass a Custom TRM

How to Pass a Custom TRM

The following code example demonstrates passing a custom transaction request message (TRM).

```
Private Sub Command1_Click()
Dim Context() As Variant
Dim COMTIContext As Object
Dim CedarBank As New COMTI.TestContext
Dim Balance As Currency
Dim TRMIN As COMTI.TRMINTestTRM
Dim TRMOUT As COMTI.TRMOUTTestTRM
Dim COMMA As Byte

TRMIN.TranID = "TCCB"
TRMIN.CommaDelim = ","
TRMIN.CommAreaLen = 41
TRMIN.ProgName = "        "
TRMIN.User = "        "
TRMIN.PSWD = "        "

Set COMTIContext = CreateObject("COMTI.ContextObject")

COMTIContext.WriteContext "TRMINTestTRM", TRMIN, Context


COMTIContext.WriteContext "TRMOUTTestTRM", TRMOUT, Context

 Call using optional context data
CedarBank.cedrbank "Name", "ID1234", Balance, Context

 Call without optional context data
CedarBank.cedrbank "Name", "ID1234", Balance


End Sub
```

See Also

Reference

ClearAllContext
CountContext
DeleteContext
QueryContextInfo
ReadContext
WriteContext

Concepts

Using Custom TRMs and ELMs with COMTIContext

Host and Automation Data

The topics in this section cover issues around working with Windows-based Automation data and COBOL-based mainframe data.

In This Section

Variably Sized Data

A string or array parameter that is the last parameter in its direction (last of the inputs and in/outs or last of the outputs and in/outs) can be smaller than the maximum size specified, even without an associated actual size. The variable string or array must be a parameter or return value and cannot be contained in a recordset.

If the final field is an array, it can be an array of any type, including a recordset. If there is an unbounded output recordset, it is the only variably-sized data allowed because Transaction Integrator (TI) cannot handle two pieces of the data stream that are variable in size. If COBOL FILLER exists after the last parameter, that parameter cannot be variable in size.

Identify the possibility of variably sized data at design time in TI Project. In the Designer select the method whose parameter list contains the final field of string to be of variable size, and set the method property "Variable Sized Final Field" to true.

See Also

Concepts

Sending Binary Data to the Host

Other Resources

Host and Automation Data

Date and Time Parameters

Transaction Integrator (TI) converts and formats the **Date** and **Time** parameters exchanged with the host differently, depending on the programming language and the host platform.

You can use TI Project to set or change the properties of the **Date** parameter. The following table shows the formatting and valid separators for each **Host Data Type** in situations where the **Data Type** property of the parameter is set to **Date**.

Data Type Formats and Separators

Host Data Type	Format (default separator)	Valid separators	Length	Notes
DATE and TIME	yyyydddhhmmsss (two packed decimal fields)	None	8	None
DATE only (COBOL only)	yyyyddd (packed decimal)	None	4	(1) (2)
DATE only (RPG only *MDY)	mm/dd/yy	/-.,&	8	(5)
DATE only (RPG only *DMY)	dd/mm/yy	/-.,&	8	(5)
DATE only (RPG only *YMD)	yy/mm/dd	/-.,&	8	(5)
DATE only (RPG only *JUL)	yy/ddd	/-.,&	6	(5)
DATE only (RPG only *LONGJUL)	yyyy/ddd	None	8	None
TIME only (COBOL only)	hhmmsss (packed decimal)	None	4	(3) (4)
TIME only (RPG only *HMS)	hh:mm:ss	:.,&	8	None
ISO DATE and TIME	yyyy-mm-dd hh.mm.ss	space	19	None
ISO DATE only	yyyy-mm-dd	-	10	None
ISO TIME only	hh.mm.ss	.	8	None
USA DATE and TIME	mm/dd/yyyy hh:mm AM (or PM)	space	19	None
USA DATE only	mm/dd/yyyy	/	10	None
USA TIME only	hh:mm AM or hh:mm PM	:	8	None
JIS DATE and TIME	yyyy-mm-dd hh:mm:ss	space	19	None
JIS DATE only	yyyy-mm-dd	-	10	None
JIS TIME only	hh:mm:ss	:	8	None
EUR DATE and TIME	dd.mm.yyyy hh.mm.ss	space	19	None
EUR DATE only	dd.mm.yyyy	.	10	None
EUR TIME only	hh.mm.ss	.	8	None
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.mmmm (length 26).	0001-01-01-00.00.00.000000	0001-01-01-00.00.00.000000	None

Where:

ISO = International Standards Organization

USA = IBM USA Standard

EUR = IBM European Standard

JIS = Japanese Industrial Standard Christian Era

Note
When a date is sent to the host, the host populates a seven-digit COMP-3 data type only with the Julian Date YYYYDDD and no other format.

Note
When a date is received from the host, the **Date** parameter must be packed as a valid Julian Date within a seven-digit COMP-3 data type.

Note
When a time is sent to the host, the host populates a seven-digit COMP-3 data type as HHMMSSS up to 100th of a second. For example, sending 01:12:03 AM populates the COMP-3 data type on the host with 0112030; sending 01:12:003 AM populates the COMP-3 data type on the host with 0112003.

Note
When a time is received from the host, the **Time** parameter must be packed within a seven-digit COMP-3 data type packed as HHMMSSS; data passed under any other format might not return the expected results.

Note
A two-digit year (yy) returned from the host is mapped to a four-digit year (yyyy) as follows:

00 to 39 is mapped as 20xx.

40 to 99 is mapped as 19xx.

Rounding occurs when TI receives the parameter from the host:

- The hour value of time rounds up the day of date.
- The minutes of time rounds up the hour of time.
- The first two digits of seconds influences the value of minutes.
- The third digit of second, or the one 1\100 value of seconds, does not influence the value of minutes. It would just be passed forward to the workstation and displayed.

For example:

- Assigning 1997001 to the host date field and 3701000 to the time field causes the workstation to display 01/02/1997 11:01:00 PM.
- Assigning 1197001 to the host date field and 0101610 to the time field causes the workstation to display 01/01/1997 01:02:01.
- Assigning 1197001 to the host date field and 0101619 to the time field, causes the workstation to display 01/01/1997 01:02:019.

Parameter Requirements

Requirements for the In, In/Out, and Out parameters might affect how you define a Transaction Integrator (TI) component or mainframe transaction program (TP). The In and In/Out parameters are sent to the mainframe-based TP from the TI Automation server. The Out and In/Out parameters are sent from the mainframe-based TP to the TI Automation server.

Best Parameter Order

The way parameters are ordered with regard to inputs and outputs determines the amount of data that must be transmitted, as well as the structure of the mainframe program. If you are creating a Transaction Integrator (TI) component in TI Project without importing COBOL code from a mainframe transaction program, place the parameters in the following order to minimize the amount of data transmitted:

- Input parameters
- Input/output parameters
- Output parameters

However, if you are using a CICS LINK LU 6.2, TCP TRM Link, or TCP ELM Link remote environment (RE) and importing COBOL data declarations into TI Project from an existing mainframe program, place the parameters in the order they appear in the COBOL data structure. In such a case, although the parameters are contained within the COMMAREA data structure, only the part of the COMMAREA containing the last input or input/output parameter is sent to the mainframe. The mainframe program is not affected by this ordering, but less data will be transmitted, especially in the case of small amounts of input data.

Maximum Amount of Parameter Data

The remote environment (RE) you use can affect the maximum possible message size. Programs associated with the CICS LINK LU 6.2, TCP TRM Link, or TCP ELM Link REs are limited by the maximum size of the COMMAREA (32,767 bytes). Therefore, the total byte size of all parameters cannot exceed 32 KB. Programs associated with distributed program call (DPC) are limited to a maximum of 65,500 bytes of user data. This maximum decreases as additional parameters are defined. DPC is limited to a maximum of 35 parameters.

The IMS Using LU 6.2 and CICS Using LU 6.2 REs have message size restrictions that, if exceeded, affect the programming logic in the mainframe program. Therefore, be careful not to exceed the limit if you use either of these REs.

Optional Metadata

As a developer, you can choose to have the Transaction Integrator (TI) run-time environment send and receive metadata to and from the mainframe transaction program (TP), and you can choose the content of that metadata.

Note
Metadata is not supported for distributed program call (DPC).

You can send or receive:

- No metadata.
- Only the method name as metadata.
- All metadata including the method name.

The TI run-time environment sends or receives metadata to or from the TP as instructed. Metadata assists the TP in:

- Identifying the format of the metadata (version information).
- Identifying the name of the method used to invoke the TP.
- Reporting detailed error information back to the client.

The metadata is not visible to the Automation client. The metadata is delivered to (or received from) the host TP as part of the request message sent to (or response message received from) the TP.

The metadata set includes the following data:

- TI run-time version.

  A string of characters, such as "Microsoft TI version 1.0.0," that uniquely identifies the TI run-time environment version that generated the request.

- Method name (32-character string) invoked by the client application code.
- Metadata block ID.

  A GUID, in character format, that uniquely identifies this block of exception data. The GUID supports the ability to have additional exception formats in the future and helps to ensure that any data received is valid.

- Variables with no assigned uses to date (reserved):
  - Boolean flag indicating whether the TP is ready to commit.
  - Boolean flag indicating whether the TP is ready to perform additional work.
  - Two Short Integers to hold pieces of the TI run-time environment version number, one Short Integer to hold the major version number and the other to hold the minor version number.
- Exception Block (used only in replies).

  A GUID, in binary format, that uniquely identifies this block of exception data. The GUID allows support of additional exception formats in the future and helps ensure the data received is valid:

- Boolean flag that indicates whether the TP is ready to commit.
- Boolean flag that indicates whether the TP is ready to perform additional work.
- Boolean flag that indicates whether an exception should be returned to the client application. If set, this flag also causes the transaction to quit.
- 16-bit integer that identifies the error (see the note later in this topic). You can assign this value, along with the 256-character message that describes the error, from the server so that the assigned value is returned when a TI run-time error occurs.
- 32-bit integer that identifies the context ID in the TP Help file (if any).
- 256-character message that describes the error. You can assign this value, along with the 16-bit integer that identifies the error from the server, so that the assigned value is returned when a TI run-time error occurs.

Metadata is always located at the beginning of the message.

Note

TI error messages have numbers in the range from 0 through 9999. Metadata error message numbers returned from the ma inframe can fall within the same range. To distinguish TI error messages from metadata messages returned from the mainfra me, TI adds 10000 to the number of any metadata error message returned from the mainframe.

Return Value Positioning

When an Automation method returns control to the calling application, it can return data as the method's value (as distinct from returning data as an output parameter). However, there is no analogous concept when you are dealing with a COBOL or Report Program Generator (RPG) data declaration.

Transaction Integrator (TI) allows you to select one of the data description entries in a data declaration that will be returned to the calling application. When you select an entry as the return value and that entry is not the first entry in the data declaration, the return value is said to be positioned after the parameters.

You can use this feature, for example, when your data declaration describes a table and you need to return a recordset on the Automation side. For example, if you are using Remote Data Service (RDS) to bind to a grid control for a Web application, your Automation method must return the recordset rather than define a parameter that represents an output recordset.

When you import host definitions, the Import COBOLor Import RPG Wizard provides a step that allows you to select data description entries as return values. If you are manually creating a method and you want the method's return value to be placed in a location other than at the front of the data declaration, you can specify the location on the **Advanced** tab of the method properties. Use the **Position return value after the parameter** drop-down list.

Data Transfer Options

You can use various options to transfer data to and from the mainframe transaction program (TP). These options apply to arrays, recordsets, and strings (or variable-length tables and display data in COBOL terms). The topics in this section describe each of the possible data transfer options. Choose the option that optimizes the amount of data to be transferred.

In This Section

- Using the OCCURS DEPENDING Clause to Define Variable-length Table
- Using Variably Sized Strings
- Using Variably Sized Rows
- Using Bounded Final Fields

Using the OCCURS DEPENDING Clause to Define Variable-length Table

In COBOL, you can use the OCCURS DEPENDING ON syntax to define a variable-length table in a data declaration. The storage for a variable-length table is dynamic, depending on the value in the length specifier variable. The amount of data passed is also dependent on the value in the length specifier variable: Only the number of elements specified are sent or received. The length specifier variable for a variable-length table must be a numeric type, and its direction must match the direction of the variable-length table it controls.

When you import COBOL into Transaction Integrator (TI) Project, and you specify variable-length tables as recordsets, the variable-length tables automatically become arrays or recordset objects whose size is limited by another parameter. The length specifier is exposed on the Automation side as a parameter and must be correctly set when the parameter is being sent to the host application.

To manually indicate that a parameter in a method is the length specifier for an array, first define the length specifier parameter, and then define the array or recordset parameter:

In the parameter property class to be defined as an ODO array, use the Designer to select the **Is Array** property. After **IsArray** is selected, the **Array Dimensions** and **Occurs depending on** property becomes available. Define the array's dimensions using the **Array Dimensions** property. Assign the ODO index to the parameter defined as the ODO array. Select the index by expanding the property **Occurs depending on**.

You can also manually indicate that a parameter in a method is the length specifier for a recordset parameter.

Follow the same steps as defined earlier; however, change the data type of the parameter from a simple data type to a recordset.

The following COBOL code shows a variable-length table:

```
01 CUSTOMER-DATA.
   05 CUSTOMER-NUMBER                PIC 9(9).
   05 LAST-NAME                      PIC X(20).
   05 FIRST-NAME                     PIC X(20).
   05 INVOICE-COUNT                  PIC 9(7) COMP-3.
   05 INVOICES OCCURS 50 TIMES DEPENDING ON INVOICE-COUNT.
      10 INVOICE-NUMBER              PIC 9(10).
      10 INVOICE-DATE                PIC 9(7) COMP-3.
      10 INVOICE-AMOUNT              PIC S9(13)V9(2) COMP-3.
      10 INVOICE-DESCRIPTION         PIC X(40).
```

The following is the method that is created when the previous COBOL is imported:

```
SendInvoices(lCustomerNo As Long, strLastName As String, strFirstName As String _
    , lcInvoices As Long) As Object
```

The following is an example of Microsoft® Visual Basic® code that calls an imported method:

```
Dim objCustomer As Object   'Uses late binding
Dim objInvoices As ADODB.Recordset
Dim lCustomerNumber As Long
Dim iRow As Integer
Dim iCol As Integer
Dim strLastName As String
Dim strFirstName As String

'create an instance of the invoicing object
On Error GoTo ErrorHandler1
Set objCustomer = CreateObject("Customer.Invoicing.1")

lCustomerNumber = CLng(txtCustomerNumber)
```

```
'invoke the GetInvoices method
On Error GoTo ErrorHandler2
Set objInvoices = objCustomer.GetInvoices(lCustomerNumber _
    , strLastName, strFirstName)
'
' Transfer the Recordset data to a variant array in a single operation.
' This is efficient, but may not be suitable for larger Recordsets.
'
Dim Data As Variant

Data = objInvoices.GetRows
grdInvoices.Rows = UBound(Data, 2) + 1
grdInvoices.Cols = UBound(Data, 1) + 1
```

Using Variably Sized Strings

When the last input parameter or the last output parameter in a method is a string, that string can be variably sized. Its size can vary from 0 to the maximum number of bytes specified for its length. When the return value is a string and it is positioned after all other output parameters, it can be the variably sized final output field.

The string must be sent or received last if it is to be variably sized. Otherwise, there is no reliable way to determine the end of a variably sized string and the next data item in the buffer. The logic of the host application sends only the data for the part of the string that is needed.

COBOL never sets the variably sized option for strings. To set this property manually, set the **Variable Sized Final Field** property to be variable. The property **Variable Sized Final Field** is subdivided into two parts by direction. Set the direction you want to true.

The following COBOL example has as its last data item a large string that could be optimized by sending only the size of the string:

```
01 CUSTOMER-DATA.
   05 CUST-HEADER.
      10 CUSTOMER-NUMBER             PIC 9(9).
      10 LAST-NAME                   PIC X(20).
      10 FIRST-NAME                  PIC X(20).
   05 COMMENTS                       PIC X(4096).
```

When imported, this COBOL code creates the following method:

```
CustomerInformation(lCustomerNo As Long,_
                    strLastName As String,_
                    strFirstName As String,-
                    strComments As String)
```

The following Visual Basic code calls the method:

```
Dim objCustomer As Object
    Dim lCustomerNo As Long
    Dim strLastName As String
    Dim strFirstName As String
    Dim strComments As String

    lCustomerNo = 100231

    'create an instance of the invoicing object
    On Error GoTo ErrorHandler1
    Set objCustomer = CreateObject("Customer.Invoicing.1")

    'invoke the SetInvoices method
    On Error GoTo ErrorHandler2
    objCustomer.CustomerInformation lCustomerNo, strLastName _
        , strFirstName, strComments
```

Using Variably Sized Rows

When the last column in a record is a string, the row can be variably sized. Its size can vary between zero and the maximum size specified in the picture clause. When you have variably sized rows, your application must explicitly specify the size of each row before the row is sent.

The actual size field is not visible on the Automation side. The Transaction Integrator (TI) run-time environment uses Automation services to determine the size of input data. When the TI run-time environment sends data to the host, it automatically sets the actual size field.

The Import COBOL Wizard never creates a recordset that has variably sized rows. Bring up properties for the recordset that contains the variably sized rows. The **Variable sized rows** property allows user to manually configure this option for a specific recordset. The **Variable sized rows** property offers advanced options. You can specify that the actual row size variable is a half-word or full-word binary. The actual size variable will include itself or will only include the size of the row.

The following COBOL example shows how the host application sends variably sized rows. The length field is included in the row size:

```
01  CUSTOMER-DATA.
    05  CUSTOMER-NUMBER                PIC 9(9).
    05  LAST-NAME                      PIC X(20).
    05  FIRST-NAME                     PIC X(20).
    05  INVOICE-COUNT                  PIC 9(7) COMP-3.
    05  INVOICES OCCURS 50 TIMES DEPENDING ON INVOICE-COUNT.
        10  INVOICE-DATA.
            15  INVOICE-ROW-SIZE       PIC S9(4) COMP.
             15  INVOICE-NUMBER         PIC 9(10).
             15  INVOICE-DATE           PIC 9(7) COMP-3.
             15  INVOICE-AMOUNT         PIC S9(13)V9(2) COMP-3.
         10  INVOICE-DESCRIPTION        PIC X(4096).
.
.
.
           MOVE LENGTH OF CUSTOMER-DATA TO SEND-LENGTH.
           SUBTRACT LENGTH OF INVOICES FROM SEND-LENGTH.
           EXEC-CICS SEND FROM(CUSTOMER-DATA)
                          LENGTH(SEND-LENGTH)
                          END-EXEC.

           PERFORM VARYING ROW FROM 1 BY 1 UNTIL ROW > INVOICE-COUNT
               INSPECT INVOICE-DESCRIPTION TALLYING INVOICE-ROW-SIZE
                   FOR CHARACTERS BEFORE INITIAL '   '
               ADD LENGTH OF INVOICE-DATA TO INVOICE-ROW-SIZE
               EXEC-CICS SEND FROM(INVOICE-ROW-SIZE)
                              LENGTH(2)
                              END-EXEC
               EXEC-CICS SEND FROM(INVOICES(ROW))
                              LENGTH(INVOICE-ROW-SIZE)
                              END-EXEC
           END-PERFORM.
```

The following COBOL example shows how the host application sends variably-sized rows. The length field is not included in the row size:

```
       01  CUSTOMER-DATA.
          05  CUSTOMER-NUMBER                PIC 9(9).
          05  LAST-NAME                      PIC X(20).
          05  FIRST-NAME                     PIC X(20).
          05  INVOICE-COUNT                  PIC 9(7) COMP-3.
          05  INVOICE-ROW-SIZE               PIC S9(4) COMP.
         05  INVOICES OCCURS 50 TIMES DEPENDING ON INVOICE-COUNT.
             10  INVOICE-DATA.
                  15  INVOICE-NUMBER         PIC 9(10).
                  15  INVOICE-DATE           PIC 9(7) COMP-3.
```

```
                15  INVOICE-AMOUNT           PIC S9(13)V9(2) COMP-3.
              10  INVOICE-DESCRIPTION         PIC X(4096).
.
.
.
           MOVE SIZE OF CUSTOMER-DATA TO SEND-LENGTH.
           SUBTRACT LENGTH OF INVOICES FROM SEND-LENGTH.
           SUBTRACT LENGTH OF INVOICE-ROW-SIZE FROM SEND-LENGTH.
           EXEC-CICS SEND FROM(CUSTOMER-DATA)
                          LENGTH(SEND-LENGTH)
                          END-EXEC.

           PERFORM VARYING ROW FROM 1 BY 1 UNTIL ROW > INVOICE-COUNT
               INSPECT COMMENTS TALLYING INVOICE-ROW-SIZE
                   FOR CHARACTERS BEFORE INITIAL '   '
               ADD LENGTH OF INVOICE-DATA TO INVOICE-ROW-SIZE
               EXEC-CICS SEND FROM(INVOICE-ROW-SIZE)
                              LENGTH(LENGTH OF INVOIVE-ROW-SIZE)
                              END-EXEC

               EXEC-CICS SEND FROM(INVOICES(ROW))
                              LENGTH(INVOICE-ROW-SIZE)
                              END-EXEC
           END-PERFORM.
```

See Also

Tasks

Using the OCCURS DEPENDING Clause to Define Variable-length Table

Using Variably Sized Strings

Using Bounded Final Fields

Using Bounded Final Fields

When the last input or the last output parameter in a method is an array or recordset, that parameter can be bounded. Its size can vary from 0 to the maximum number of elements or rows specified. The array or recordset must be last to be bounded. Otherwise, there is no reliable way to determine the end of a bounded array or recordset and the beginning of the next field in the buffer. The host application must take care of sending the truncated table.

The Automation client handles this option automatically. The Transaction Integrator (TI) run-time environment sends a truncated amount of data based on the Automation bounds and detects the truncated data and creates the appropriate Automation type when data is received.

The Import COBOL Wizard never sets the bounded option for arrays or recordsets. To set this manually for the final parameter in a method, use the Designer to assign a value to the property **Maximum Occurrence**. This field defines the maximum number of rows the recordset may contain. On the method containing the recordset, set the property **Variable Sized Final Field** to true by direction to make the recordset bounded.

If the method contains a recordset that is unbounded, you cannot also specify a bounded or variably-sized final field for that direction. For example, if Parameter1 is an output parameter, and it is an unbounded recordset, the final output parameter cannot be a bounded array or recordset or variably-sized string. When the return value is positioned after all other output parameters, the return value can be the bounded final output field.

The following COBOL example sends only some of the rows in a recordset:

```
01  INVOICE-COUNT                        PIC S9(4) COMP.
01  CUSTOMER-DATA.
    05  CUSTOMER-NUMBER                  PIC 9(9).
    05  LAST-NAME                        PIC X(20).
    05 INVOICES OCCURS 50 TIMES.
               10  INVOICE-NUMBER              PIC 9(10).
               10  INVOICE-DATE                PIC 9(7) COMP-3.
               10  INVOICE-AMOUNT              PIC S9(13)V9(2) COMP-3.
.
.
.
            MOVE SIZE OF CUSTOMER-DATA TO SEND-LENGTH.
            SUBTRACT LENGTH OF INVOICES FROM SEND-LENGTH.
            EXEC-CICS SEND FROM(CUSTOMER-DATA)
                           LENGTH(SEND-LENGTH)
                           END-EXEC.
            PERFORM VARYING I FROM 1 BY 1 UNTIL I = INVOICE-COUNT
                COMPUTE SEND-LENGTH = LENGTH OF INVOICE-NUMBER +
                                      LENGTH OF INVOICE-DATE +
                                      LENGTH OF INVOICE-AMOUNT
                EXEC-CICS SEND FROM(INVOICES(I))
                               LENGTH(SEND-LENGTH)
                               END-EXEC.
            END-PERFORM.
```

Mainframe Character Strings and Code Pages

When Transaction Integrator (TI) sends data to a mainframe-based transaction program (TP), the TI run-time environment transforms Unicode strings received as parameters, fields, or columns into mainframe character strings. Likewise, when it receives data from a mainframe TP, the TI run-time environment converts the mainframe character strings into Unicode strings to be returned as output values to the calling client application.

TI categorizes these strings of characters sent to and received from the mainframe as follows:

- Extended Binary Coded Decimal Interchange Code (EBCDIC) strings.
- IBM double-byte character set (DBCS) strings.
- Intermixed strings containing both EBCDIC and IBM DBCS strings with the necessary shift-out (SO) and shift-in (SI) characters.

The TI run-time environment determines the type of mainframe character string based on the following information:

- How the parameter, field, or column is defined in the TI component that was built by using TI Project.
- The code page defined for the specific remote environment (RE) that was associated with the active TI Automation server when it was deployed. When you create an RE in TI Manager, you specify a code page for that RE.

In This Section

How to Assign a Different Code Page to a Remote Environment

Use a remote environment (RE) to describe each CICS or IMS mainframe environment. Use Transaction Integrator (TI) Manager to create, modify, and delete REs.

Every RE description includes a property value for the code page that is to be used by the TI run-time environment to convert characters to and from mainframe data representations.

When you create a new RE, TI Manager automatically selects a code page based on the current system locale. However, you can replace this default value with a code page that you select.

As needed, you can use TI Manager to assign a different code page to any RE.

To select a different code page for an RE

1. On the **Start** menu, point to **Programs**, then **Microsoft Host Integration Server 2009**, and then click **TI Manager**.
2. Expand the **Remote Environments** node.
3. Right-click the remote environment to be modified, and then click **Properties**.
4. On the **Locale** tab, clear the check box **Use default code page for the selected locale**.
5. On the **Locale** tab, select the new code page from the **Code page** dropdown list.

Only the code page value is significant to the TI run-time environment when it converts character data. Although you can use the locale displayed on the **Locale** property page to select the code page, the TI run-time environment ignores it and uses only the code page value that is associated with that locale.

See Also

Other Resources

Mainframe Character Strings and Code Pages

IBM DBCS Code Pages

Transaction Integrator (TI) recognizes the following code pages as IBM double-byte character set (DBCS) code pages:

Code page	Description
930	IBM Extend Katakana and IBM Japanese (for Japan)
931	IBM English lower case and IBM Japanese (for Japan)
933	IBM Hungle Extend single-byte and IBM Hungle (for Korea)
935	IBM English single-byte and IBM simplified Chinese (for PRC)
937	IBM English single-byte and IBM traditional Chinese
939	IBM English Extend lower case and IBM Japanese (for Japan)

All other code pages delivered with Host Integration Server 2009 are handled as Extended Binary Coded Decimal Interchange Code (EBCDIC) code pages, which use a single byte to represent a character.

When the TI run-time environment converts UNICODE characters to DBCS characters, it uses standard Microsoft Windows NLS code pages.

The following table lists the DBCS code pages and their corresponding NLS code pages.

DBCS code page	NLS code page required by TI
930	932 ANSI Japan
931	932 ANSI Japan
933	949 ANSI Korean
935	936 ANSI Chinese (China, Singapore)
937	950 ANSI Chinese
939	932 ANSI Japan

The list of code pages on the **Locale** property page for a remote environment includes a DBCS code page only if the corresponding NLS code page is loaded on the computer. The following code pages have been added for European support.

Code page	IBM or open systems name	Display name
923	ISO 8859-15 Latin 9 (Euro)	Latin-9 (Euro)
924	IBM Latin - 1/Open System	Latin-1 Open System (Euro)
858	OEM - Multilingual Latin 1 (Euro)	OEM - Multilingual Latin 1 (Euro)
1140	IBM EBCDIC - U.S./Canada (Euro)	U.S./Canada (Euro)
1141	IBM EBCDIC - Germany (Euro)	Germany (Euro)
1142	IBM EBCDIC - Denmark/Norway (Euro)	Denmark/Norway (Euro)

See Also

Other Resources

Mainframe Character Formats

In Transaction Integrator (TI) Project, you can specify the mainframe character format that the TI run-time environment will create when sending data to the mainframe. There are two mainframe character formats supported by TI:

- PIC X(n) COBOL, or RPG A
- PIC G(n) COBOL, or RPG G

When you create string parameters, fields, or columns in TI Project, the PIC X(*n*) or RPG A data type format is selected automatically.

If necessary, you can use the **Properties** command to change the mainframe character format.

If you select the PIC X or RPG A format for a string, the TI run-time environment converts this string to either an Extended Binary Coded Decimal Interchange Code (EBCDIC) character string or an intermixed character string. Specifically, if the TI component you define in TI Project is assigned to a remote environment (RE) with an EBCDIC code page, the TI run-time environment converts a string that has a PIC X or RPG A format into an EBCDIC string. If the TI component's RE identifies a double-byte character set (DBCS) code page, the TI run-time environment converts a string that has a PIC X format as an intermixed string (not supported for RPG).

If you select the PIC G or RPG G format for a string, the TI run-time environment always converts the string into a DBCS string. Therefore, any TI component that uses a string with a PIC G or RPG G format must be assigned to an RE that has a DBCS code page.

If a TI component using a string with a PIC G or RPG G format is assigned to an RE that has an EBCDIC code page, the TI run-time environment reports a conversion error when it attempts to convert the string to or from the PIC G or RPG G format. The TI run-time environment places an error message describing this conversion problem in the Windows Event Log, and it returns an error to the invoking client application.

The following table summarizes how the selection of string format and code page controls the type of character conversion performed by the TI run-time environment.

String format	EBCDIC code page	DBCS code page
PIC X or RPG A	EBCDIC string	Intermixed string
PIC G or RPG G	TI run-time environment reports conversion errors.	DBCS string

String Dimension Values

The meaning of a string's dimension (the *n* part of the PIC X(n) or RPG A(n) and the *n* part of the PIC G(n) or RPG G(n) formats) is based on the character format in use. You specify a string's dimension on the **COBOL Definition** property page in Transaction Integrator (TI) Project.

- The dimension value for a string with a PIC G or RPG G format gives the number of double-byte characters that are used in the mainframe representation of the string. No SO and SI character pair is added when a string with a PIC G or RPG G format is converted.
- The dimension value for a string with a PIC X or RPG A format gives the number of bytes that are used in its mainframe representation. The number of characters that can be placed into or taken from a PIC X or RPG A formatted string varies depending on the number of :
    - Double-byte character set (DBCS) characters, each of which requires two bytes of storage.
    - SO and SI character pairs needed. Each two-byte pair must encapsulate each contiguous stream of DBCS characters.

Developers using TI must take into account this variability in the size of an intermixed string when they specify dimension values in TI Project.

The number of bytes for a string converted using an EBCDIC code page with a PIC X or RPG A format is identical to the number

of characters because there are no DBCS characters in the string.

However, for a string converted using a DBCS code page with a PIC X or RPG A format, the actual number of characters that can be placed in a given number of bytes varies. For example, if the conversion to or from UNICODE does not require the use of DBCS characters (that is, no SO and SI character is used in the mainframe string), each character occupies a single byte. However, if DBCS characters do appear within the mainframe string, the SO and SI character pairs are needed.

How the Import Wizard Defines Strings

When you use Transaction Integrator (TI) Project's Import wizard to import a host definition to create new methods and recordsets, the wizard selects the mainframe character format based on the imported host definition. The following table shows how the wizard maps different COBOL declarations to a string.

COBOL type	**Type of string created**
PIC X(*n*) or RPG A	String of size *n* bytes
PIC G(*n*) or RPG G	String of size *n* characters

See Also

Other Resources

Mainframe Character Strings and Code Pages

How to Pad Mainframe Character Strings with Spaces

You can define the properties for a string such that the Transaction Integrator (TI) run-time environment adds space characters to pad the mainframe representation of the string instead of depending on a null termination character.

To use either a space character or null termination character

1. In Microsoft Visual Studio, right-click the object, and then click **Properties**.
2. Under **Host Data Type Information** in the **Properties** pane, click **String Delimiting**.
3. Select either **Space Padded** or **Null Terminated**.

The following table describes what happens with each delimiting option (**Space Padded** or **Null terminated**) when converting to the type of string indicated.

Type of string operation	**What happens for each type of string delimiting operation**
Conversion to EBCDIC string	**Space Padded.** The TI run-time environment adds single-byte space characters to the end of the string until all bytes in the PIC X formatted string are filled.
	Null terminated. The TI run-time environment adds a single null character to the end of the string if there is room in the PIC X count for the byte.
Conversion from EBCDIC string	**Space Padded.** The TI run-time environment strips single-byte space characters from the end of the string.
	Null terminated. The TI run-time environment scans from the beginning of the string and stops the conversion at the first null character it encounters in the string.
Conversion to DBCS string	**Space Padded.** The TI run-time environment adds double-byte space characters to the end of the string until all characters in the PIC G formatted string are filled.
	Null terminated. The TI run-time environment adds a double-byte character set (DBCS) null character to the end of the string if there is room in the PIC G count for the bytes.
Conversion from DBCS string	**Space Padded.** The TI run-time environment strips double-byte space characters from the end of the string.
	Null terminated. The TI run-time environment scans from the beginning of the string and stops the conversion at the first DBCS null character it encounters in the string.
Conversion to Intermixed string	**Space Padded.** The TI run-time environment adds single-byte space characters to the end of the string until all bytes in the PIC X formatted string are filled. If the terminating character in the UNICODE string maps to a DBCS character, the TI run-time environment adds an SI character before it adds the space characters.
	Null terminated. The TI run-time environment adds a single byte null character to the end of the string if there is room in the PIC X count. If the terminating character in the UNICODE string maps to a DBCS character, the TI run-time environment adds an SI character before it adds the null character.

Conversion from Intermixed string	**Space Padded.** The TI run-time environment strips the terminating single-byte and double-byte space characters from the end of the string. When it strips the space characters, the TI run-time environment treats any terminating SI character as if it were a space.
	Null terminated. The TI run-time environment scans from the beginning of the string and stops the conversion at the first null character (of either width) it encounters.

Special handling occurs for a string that is last in the host buffer and that is flagged as *last is variable*. For example:

- **Space Padded.** Upon conversion to an Extended Binary Coded Decimal Interchange Code (EBCDIC) string, the string is terminated by the length count of the containing buffer, so it contains no additional space characters. Upon conversion from an EBCDIC string, the buffer is considered terminated by the length count of the containing buffer; then the string is examined for blank padding. The host can send this string blank padded beyond the significant data or not blank padded but with the last significant character of the string in the last position in the containing buffer. The space character is determined by the type of string (single, double, or intermixed).

- **Null terminated.** Upon conversion to an EBCDIC string, the string is sent as is. The TI run-time environment checks the length of the string, and then checks that the exact number of characters is sent. In other words, the number of characters sent is equal to the length of the string. No null terminator or spaces are appended to the end of the string.

The following tables show how string delimiting works when the **String delimiting** property is set to **Space Padded** versus **Null terminated** in combination with the variable size setting. All examples assume the mainframe data declaration as PIC X(5). "b" represents a space,"?" represents unassigned data, and "\0" represents a null.

String delimiting set to Space Padded and variable size not active

Workstation	Direction	Mainframe
ABC\0	To Host	'ABCbb'
ABCb	To Host	'ABCbb'
CBA	From Host	'CBAbb'
CBA\0?	From Host	'CBA\0?'
CBA\0	From Host	'CBA\0b'

String delimiting set to Space Padded and variable size active

Workstation	Direction	Mainframe
ABC\0	To Host	'ABC'
Abb	To Host	'Abb'
CBA	From Host	'CBAbb'
CBA\0?	From Host	'CBA\0?'
CBA\0	From Host	'CBA\0b'

String delimiting set to Null terminated and variable size not active

Workstation	Direction	Mainframe
ABC\0	To Host	'ABC\0?'
ABC	From Host	'ABC\0?'
ABCbb	From Host	'ABCbb'

ABC	From Host	ABC\0\0'

String delimiting set to Null terminated and variable size active

Workstation	Direction	Mainframe
ABC\0	To Host	'ABC\0'
ABC	From Host	'ABC\0?'
ABCbb	From Host	'ABCbb'
ABC	From Host	ABC\0\0'

See Also

Other Resources

Mainframe Character Strings and Code Pages

Truncating Undefined Portions of Strings

You can define the properties for a string such that the Transaction Integrator (TI) run-time environment truncates undefined characters when it converts UNICODE strings to mainframe data representations instead of generating an error message. To do so, click **Truncate** under **Error handling** on the string's **Host Definition** tab (property page) in TI Project.

When truncation is turned on, the TI run-time environment limits the number of characters to the string's previously specified dimension value when it converts a character string to an Extended Binary Coded Decimal Interchange Code (EBCDIC) or double-byte character set (DBCS) character string.

When it converts to intermixed strings, the TI run-time environment ensures that all shift-out (SO) characters have matching shift-in (SI) characters. It adds a terminating SI character when truncation occurs in the middle of a contiguous stream of DBCS characters. Also, the TI run-time environment ensures that it does not leave a partial DBCS character when it adds the SI character.

If blank padding and truncation are specified for a string, the TI run-time environment might need to add an EBCDIC space character after a terminating SI character is added.

See Also

Reference

Converting Data Types from Automation to OS/390 COBOL

Converting Data Types from OS/390 COBOL to Automation

Adding Leading SO and Trailing SI Characters

For PIC G formatted strings, you can instruct the Transaction Integrator (TI) run-time environment to add a leading shift-out (SO) character and a trailing shift-in (SI) character by selecting the **Add Leading SO and Trailing SI** check box on the string's **COBOL Definition** tab (property page) in TI Project.

If the **Add Leading SO and Trailing SI** check box is selected, the TI run-time environment handles two additional bytes in the mainframe data structure used for describing the double-byte character set (DBCS) string. When it formats a message sent to the mainframe, the TI run-time environment adds the leading SO and trailing SI bytes. When it interprets a message received from the mainframe, the TI run-time environment discards the leading SO and the trailing SI bytes.

The dimension value of the PIC G string always specifies the number of double-byte DBCS characters in the strings, regardless of the presence or absence of the surrounding SO and SI characters.

The use of this automatic SO and SI handling is hidden from the client application. However, the mainframe application must ensure that the appropriate PIC X declarations surround the declaration of the PIC G string.

TI Project generates the appropriate declarations for the surrounding SO and SI bytes, as shown in the following sample code:

```
01     A-SOSI-WRAPPED-DBCS.
       05  LEADING-SO-1                          PIC X.
       05  MY-DBCS-STRING                        PIC G(80).
       05  LEADING-SI-1                          PIC X.
```

The Import COBOL Wizard in TI Project does not set the option to add leading SO and trailing SI bytes. In other words, the Import COBOL Wizard places no significance on the presence of PIC X declarations surrounding a PIC G string. If an existing mainframe transaction program (TP) uses COBOL declarations that contain explicit declarations for SO and SI characters that wrap PIC G strings, you must manually modify the interface created by the Import COBOL Wizard.

Maximum Buffer Sizes for Remote Environments

Both Transaction Integrator (TI) and mainframe applications must be designed not to exceed the following buffer size or transfer limitations:

TCP Transaction Request Message (TRM) Link

The maximum buffer size for input/output is 32767 bytes. This limit is defined by the maximum allowable size of the COMMAREA.

TCP Enhanced Listener Message (ELM) Link

The maximum buffer size for input/output is 32767 bytes. This limit is defined by the maximum allowable size of the COMMAREA.

TCP Transaction Request Message (TRM) User Data

The buffer size for input/output is unlimited.

TCP Enhanced Listener Message (ELM) User Data

The buffer size for input/output is unlimited.

IMS Connect (TCP/IP)

The maximum segment size that TI can send to IMS is 32754 bytes. This limit is defined by the total number of bytes (32767) in an IMS message segment minus the following number of bytes:

- 2 bytes for the LL (Length field).
- 2 bytes for the ZZ (Control Information field).
- 9 bytes for the maximum TRANCODE.

The total amount of data that can be sent to, or received from, an IMS server program using the TCP/IP IMS Connect programming model is unlimited.

IMS Implicit (TCP/IP)

The buffer size for input is from 1 through 32755 bytes. In other words, the IMS Assist Module rejects a request with zero bytes of user data, and the IMS Listener rejects a request with more than 32755 bytes of user data.

The buffer size for output is from 0 through 32755 bytes. The IBM IMS default Listener enforces the limit of 32755 bytes of input/output data. The IBM IMS Assist Module enforces the required minimum of 1 byte of input.

IMS Explicit (TCP/IP)

The buffer size for input/output is unlimited.

OS/400 Distributed Program Calls (DPC)

The total buffer size is 65535 bytes and is reduced by the required headers. The send requires 23 bytes of header.

Each parameter, of either direction, requires 12 bytes of overhead on the send. Each in\out or out parameter requires 12 bytes of overhead within memory on the return trip.

CICS LINK LU 6.2

The maximum buffer size for input/output is 32767 bytes. This limit is defined by the maximum allowable size of the COMMAREA.

CICS User Data LU 6.2

The buffer size for input/output is unlimited. However, if the size of the input buffer is greater than 32 KB, TI sends the data in multiple chunks of 32 KB. In such a case, the mainframe application must do multiple receives to gather all the input data.

IMS Using LU 6.2

The maximum segment size that TI will send to IMS is 32754 bytes. This limit is defined by the total number of bytes (32767) in

an IMS message segment minus the following number of bytes:

- 2 bytes for the LL (Length field).
- 2 bytes for the ZZ (Control Information field).
- 9 bytes for the maximum TRANCODE.

The total amount of data that can be sent to, or received from, an IMS server program that uses LU 6.2 is unlimited.

See Also

Other Resources

Host and Automation Data

Alignment Problems with Generated COBOL

COBOL aligns data elements at the 01 level on double-word boundaries. This practice causes a potential problem in CICS non-DPL applications that use TI-generated data declarations along with error metadata. If you code your COBOL application to receive the error metadata and the input parameters in one RECEIVE, the parameters are placed immediately adjacent to the metadata in memory. However, because the error metadata does not end on a double-word boundary, this action puts the parameters 4 bytes ahead of where the COBOL code expects them.

You can prevent this problem. When you click either the **Include method name** or the **Include all information** option under **Meta data** on the **Advanced** tab of a method's property page, verify that the mainframe program issues two RECEIVE commands to handle incoming data for the method. The first RECEIVE pulls in the metadata block, and the second RECEIVE pulls in the data for the method. When COBOL is generated for the method, an additional 01 block is generated for the metadata. When the **Include all information** option is selected, you are also expected to create an additional SEND for the metadata before sending the method data back to the Automation client application.

See Also
Other Resources
Host and Automation Data

Filler

This section provides recommendations for handling COBOL-based filler data.

In This Section

COBOL FILLER

TI Application Cannot Reference FILLER

How to Use REDEFINES in COBOL

FILLER Optimization

FILLER for Discontiguous Output Area and Return Value

COBOL FILLER

If COBOL data declarations that are imported into Transaction Integrator (TI) Project contain a COBOL FILLER clause or clauses, space is allocated in the message so that the filler is represented correctly for the mainframe program's data alignment. The filler is hidden from the Automation interface.

See Also

Concepts

FILLER Optimization

Other Resources

Filler

TI Application Cannot Reference FILLER

There are at least three possible causes for why the application cannot reference FILLER data:

- Mainframe or COBOL specifics.
- Automation specifics.
- Procedure using TI Project.

The following provides details of these three causes.

Mainframe or COBOL Specifics

When a FILLER keyword is encountered in the import process, the Transaction Integrator (TI) run-time environment adjusts the offset for the position of the data that follows the filler in a send or receive buffer by the length of the filler. This leaves untranslated gaps in the buffers that are sent to (or received from) the host and allows your data to overlay correctly onto the data declaration that describes it.

Automation Specifics

The Automation method does not reference the filler data description entries.

Procedure Using TI Project

The filler that is at the start of a data declaration is associated with a method, recordset, datatable, user-defined type (UDT), or .NET structure. You can view or change filler that is associated with a method from the **Advanced** tab of the method's properties page. To view or change a filler that is associated with a method, recordset, or UDT, right-click the method, recordset, or UDT, and then click **Properties**.

Filler that follows a data description entry is associated with the data description entry (or parameter for methods, column for recordsets, or member for UDTs). You can view or change filler that is associated with a parameter, column, or member from the **COBOL Definitions** tab of the parameter, column, or member properties. When filler follows the data description entry that you have specified as the return value, you can view or change that filler from the **COBOL Definitions** tab of the method's properties.

The following example shows a COBOL data declaration that uses FILLER:

```
01 CUSTOMER-DATA.
   05 CUSTOMER-INFO.
      10 LAST-NAME               PIC X(20).
      10 FIRST-NAME              PIC X(20).
      10 FILLER                  PIC X(12).
   05 DEMOGRAPHICS.
      10 DEMO-AGE                PIC 999.
      10 DEMO-INCOME             PIC S9(9)V99 COMP-3.
      10 DEMO-SEX                PIC X.
      10 DEMO-MSTATUS            PIC X.
      10 FILLER                  PIC X(40).
```

The resulting method is:

```
CustomerDemographics(strLastName As String, strFirstName As String, iAge As Integer _
    , curIncome As Currency, strSex As String, strMStatus As String)
```

The following is an example of the Visual Basic code that calls the method:

```
Dim objCustomer As Object
    Dim strLastName As String
    Dim strFirstName As String
    Dim iAge As Integer
```

```
    Dim curIncome As Currency
    Dim strSex As String
    Dim strMStatus As String

    strLastName = "Doe"
    strFirstName = "John"

    'create an instance of the invoicing object
    On Error GoTo ErrorHandler1
    Set objCustomer = CreateObject("Customer.Invoicing.1")

    'invoke the SetInvoices method
    On Error GoTo ErrorHandler2
    objCustomer.CustomerDemographics strLastName, strFirstName _
        , iAge, curIncome, strSex, strMStatus
```

See Also

Other Resources

Filler

How to Use REDEFINES in COBOL

The COBOL import process in Transaction Integrator (TI) Project recognizes the REDEFINES clause in a data description entry and correctly associates the redefining entries with the redefined entry. You must select one of the redefined or redefining clauses as the entry that represents the data that will be transmitted.

The redefining entries can use less space than the redefined entry. If you select a redefining entry that is smaller than the redefined entry, TI Project automatically adds filler so that the data will correctly overlay the data description when it is sent to the host. If the redefining entry represents a table with multiple fields, the last field contains the filler.

The following COBOL example shows a REDEFINES clause. The redefining clause was selected during import:

```
01 CUSTOMER-DATA.
   05 CUSTOMER-ID                          PIC X(10).
   05 CUSTOMER-ID-PARTS REDEFINES CUSTOMER-ID.
      10 LOCATION                          PIC X(3).
      10 NAME-ABREV                        PIC X(5).
```

The resulting method that is imported is:

```
CreateCustomerID(strLocation As String, strNameAbrev As String)
```

The COBOL generated for this method is:

```
01 CREATECUSTOMERID-INPUT-AREA.
   05 LOCATION              PIC X(3).              INPUT
   05 NAME-ABREV            PIC X(5).              INPUT
   05 FILLER                PIC X(2).              INPUT
```

The FILLER is added to the CUSTOMER-ID redefined area. When this FILLER occurs at the end of send or receive buffers, for performance reasons it is not sent.

The following is an example of Visual Basic code that calls this method:

```
    Dim objCustomer As Object
    Dim strLocation As String
    Dim strNameAbrev As String

    strLocation = "101"
    strNameAbrev = "SPORT"

    'create an instance of the invoicing object
    On Error GoTo ErrorHandler1
    Set objCustomer = CreateObject("Customer.Invoicing.1")

    'invoke the CreateCustomerID method
    On Error GoTo ErrorHandler2
    objCustomer.CreateCustomerID strLocation, strNameAbrev
```

See Also

Other Resources

Filler

FILLER Optimization

The Transaction Integrator (TI) run-time environment optimizes buffers sent to the host by not sending the bytes corresponding to FILLER that appear at the end of the input data declaration.

In a CICS LU 6.2 environment, use both the MAXLENGTH and NOTRUNCATE options on the RECEIVE statement when the TI run-time environment receives an input area that contains FILLER as the last data item description. However, this requirement does not apply to CICS LINK and IMS transaction programs.

See Also

Other Resources

Filler

FILLER for Discontiguous Output Area and Return Value

If the return value is discontiguous from the output area, you must calculate and manually specify the filler between the return value and the output area.

The following example shows the calculation for the filler from the original COBOL that goes into the Import Wizard (the byte counts on the right are added as an illustration):

```
01  OUTPUT-AREA.
           05  SELECTED-OUTPUT-AREA.
               10  FIELD1                      PIC S9(4)       COMP.     [2 Bytes]
               10  FIELD2                      PIC S9(9)       COMP.     [4 Bytes]
           05  DISCONTIG-UNSELECTED-AREA.
               10  NOTSELECTED                 PIC X(10).                 [10 Bytes]
               10  ALSO-NOTSELECTED            PIC S9(9)       COMP.     [4 Bytes]
           05  RETVAL                          PIC S9(9)       COMP.     [4 Bytes]
```

In this case, because the return value follows the output area, filler must be added to the last output parameter. To do this, perform the following steps.

1. Unlock the method.
2. In the details pane, click **FIELD2**.
3. On the **File** menu, click **Properties**, and then click the **COBOL Definition** tab.
4. In the **From Host** box, type 14 as the trailing filler.
5. Click **OK**.

To verify your modified code, in **TI Project**, use the **Export** command on the **File** menu. You can then see your code in Notepad.

The following is the output with the added filler:

```
       01  DISCONTIGCBL-OUTPUT-AREA.
           05  LL                           PIC S9(4) COMP.         OUTPUT     [2 Bytes
]
           05  ZZ                           PIC S9(4) COMP.         OUTPUT     [2 Bytes
]
           05  FIELD1                       PIC S9(4) COMP.         OUTPUT     [2 Bytes
]
           05  FIELD2                       PIC S9(9) COMP.         OUTPUT     [4 Bytes
]

           05  RETVAL                       PIC S9(9) COMP.         OUTPUT     [4 Bytes
]
```

See Also
Other Resources
Filler

Variable-length Tables and CICS LINK

When an OCCURS clause describes a variable-length table in the CICS LINK environment, the storage that the table uses on the host varies depending on the value of the length specifier. COBOL handles this storage automatically on the host, but for Transaction Integrator (TI) to determine where in the buffer to place data sent to the host and where to unpack data from the host, you must give it the value of the length specifier variable on which the table size depends.

Any data that follows a variable-length table must be correctly offset in the buffer immediately following the table, regardless of the maximum length of the table. TI must have the length specifier value for a variable-length table both when it packs the buffer to be sent and when it unpacks the buffer that is received.

If an OCCURS clause describes a variable-length table, you must specify the table and the length specifier that controls the table length as input/output in TI Project. The TI run-time environment must be able to detect the length both when the buffer is sent to the host and when it is received from the host. When you import COBOL or manually create a method that describes a variable-length table in TI Project, this restriction is enforced.

Note
Information in this topic also applies to arrays.

See Also

Tasks

Using the OCCURS DEPENDING Clause to Define Variable-length Table

Other Resources

Arrays

Sending Binary Data to the Host

The Transaction Integrator (TI) run-time environment does not translate binary data (Byte in Visual Basic, VT_UI1 in Automation, or unsigned char in C++) when it sends it to the mainframe. Instead, TI copies the binary data unchanged into a PIC X data representation on the mainframe. To define a string of binary data, define it as an array.

See Also

Other Resources

Host and Automation Data

COMTIContext Interface

Use the **ICOMTIContext** interface to add, remove, and query information in the Transaction Integrator (TI Context). The following methods implement the **ICOMTIContext** interface.

In This Section

ClearAllContext

ClosePersistentConnection

CountContext

DeleteContext

GetConnectionInfo

QueryContextInfo

ReadContext

UpdateContextInfo

WriteContext

ClearAllContext

Use the **ClearAllContext** method to remove all the entries in a TI Context array.

Syntax

```
HRESULT ClearAllContext (
```

Parameters

prgContextArray

This SAFEARRAY parameter contains the TI Context that is to be cleared.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClosePersistentConnection
CountContext
DeleteContext
GetConnectionInfo
QueryContextInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

ClosePersistentConnection

Use the **ClosePersistentConnection** method to close the persistent connection by contacting the COM+ or .NET Framework application object without the need for a call to the server object.

Syntax

```
HRESULT ClosePersistentConnection (
```

Parameters

COMTIContextArray

This SAFEARRAY contains the state of the connection.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
CountContext
DeleteContext
GetConnectionInfo
QueryContextInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

CountContext

Use the **CountContext** method to acquire a count of the entries in a TI Context array.

Syntax

```
HRESULT CountContext (
```

Parameters

prgContextArray

This SAFEARRAY parameter contains TI Context entries that are passed to and returned from COM+ objects.

pulContextEntriesCount

Upon successful completion of the call, this integer parameter contains a count of the number of entries in the specified TI Context array.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
DeleteContext
GetConnectionInfo
QueryContextInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

DeleteContext

Use the **DeleteContext** method to remove an entry in a TI Context array.

Syntax

```
HRESULT DeleteContext (
```

Parameters

bstrContextEntryName

This string parameter contains the name of the TI Context entry to be removed.

prgContextArray

This SAFEARRAY parameter contains the TI Context entries that are passed to and returned from COM+ objects.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
GetConnectionInfo
QueryContextInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

GetConnectionInfo

Use the **GetConnectionInfo** method to query the state of the persistent connection.

Syntax

```
HRESULT GetConnectionInfo (
```

Parameters

COMTIContextArray

This SAFEARRAY contains the state of the connection.

pfConnectionIsPersistent

This BOOL parameter is set to TRUE if the connection is persistent and active.

pfConnectionIsViable

This BOOL parameter is set to TRUE if the connection is active.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
DeleteContext
QueryContextInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

QueryContextInfo

Use the **QueryContextInfo** method to acquire a count of the entries in a TI Context array, the names of the context entries, and the data types of the context entries.

Syntax

```
HRESULT QueryContextInfo (
```

Parameters

prgContextArray

This SAFEARRAY parameter contains the TI Context entries that are passed to and returned from COM+ objects.

pulContextEntriesCount

Upon successful completion of the call, this integer parameter contains a count of the entries in the specified TI Context array.

prgContextNameArray

This SAFEARRAY of strings parameter contains the TI Context entry names.

prgContextTypeArray

This SAFEARRAY of integers parameter contains the TI Context entry data types.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
DeleteContext
GetConnectionInfo
ReadContext
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

ReadContext

Use the **ReadContext** method to acquire the value of an entry in a TI Context array.

Syntax

```
HRESULT ReadContext (
```

Parameters

bstrContextEntryName

This string parameter contains the name of the TI Context entry to be retrieved.

pvarContextEntryValue

This VARIANT parameter contains the value of the context entry identified by bstrContextEntryName.

prgContextArray

This SAFEARRAY parameter contains the TI Context entries that are passed to and returned from COM+ objects.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
DeleteContext
GetConnectionInfo
QueryContextInfo
UpdateContextInfo
WriteContext

Other Resources

Using a Persistent Connection

UpdateContextInfo

Use the **UpdateContextInfo** method to update the client **COMTIContext** array with the current state of the connection.

Syntax

```
HRESULT UpdateContextInfo (
```

Parameters

COMTIContextArray

This SAFEARRAY contains the state of the connection.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
DeleteContext
GetConnectionInfo
QueryContextInfo
ReadContext
WriteContext

Other Resources

Using a Persistent Connection

WriteContext

Use the **WriteContext** function to add or replace an entry in a TI Context array.

Syntax

```
HRESULT WriteContext (
```

Parameters

bstrContextEntryName

This string parameter contains the name under which the TI Context entry will be stored and accessed. Some context entry names are predefined by Microsoft. You are free to define your own (although there is no way to deal with such in the COM+ application).

varContextEntryValue

This VARIANT parameter contains the value to be stored for the context entry identified by bstrContextEntryName.

prgContextArray

This SAFEARRAY parameter contains the TI Context entries that are passed to and returned from COM+ objects.

Return Codes

S_OK

The method call completed successfully.

E_INVALIDARG

One or more of the parameters passed are invalid.

E_FAIL

An internal processing error occurred.

See Also

Reference

ClearAllContext
ClosePersistentConnection
CountContext
DeleteContext
GetConnectionInfo
QueryContextInfo
ReadContext
UpdateContextInfo

Other Resources

Using a Persistent Connection

COMTIContext Keywords

Use the COMTIContext keywords as commands to override the contents of a transaction request message (TRM).

In This Section

CONNTIMEOUT

Use the CONNTIMEOUT keyword to reclaim orphaned persistent connections. **CONNTIMEOUT** takes an integer value specifying, in seconds, how long to wait before a persistent connection is considered abandoned and then automatically closed. The timing starts as the client call processing is completed by the COM+ or .NET generic object.

See Also

Tasks

How to Pass a Custom TRM

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMIN

Using Custom TRMs and ELMs with COMTIContext

CONNTYPE

Other Resources

ProgNameOverride

PASSWORD

CONNTYPE

Use the CONNTYPE keyword to establish a persistent connection by setting **CONNTYPE** to **OPEN**. If a call with **CONNTYPE** set to **OPEN** completes successfully, the returned **COMTIContext** array **CONNTYPE** keyword has a value of **USE**. To make a call and terminate the persistent connection, set the **CONNTYPE** keyword to **CLOSE**. If a call with **CONNTYPE** set to **CLOSE** completes successfully, the returned **COMTIContext** array **CONNTYPE** keyword has a value of **NONPERSISTENT**.

See Also

Tasks

How to Pass a Custom TRM

Concepts

TPNameOverride
USERID
IMS_LTERM
TRMIN
Using Custom TRMs and ELMs with COMTIContext
CONNTIMEOUT

Other Resources

ProgNameOverride
PASSWORD

IMS_LTERM

Use the **IMS_LTERM** to override the default LTERM that an Information Management Systems (IMS) connect call uses while processing the host transaction within IMS. The COMTIContext context name is **IMS_LTERM**, and its value must be from one through eight alphanumeric characters.

Note
If the **IMS_LTERM** is assigned more than eight characters, the **IMS_LTERM** value is not passed to the host transaction and th e host transaction continues to use the system provided default LTERM.

See Also

Tasks

How to Pass a Custom TRM

Reference

REOverride

Concepts

TPNameOverride

USERID

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

IMS_MODNAME

When using the IMS Connect programming model, IMS_MODNAME contains the name of the returned MOD Name assigned by the executed IMS transaction.

See Also

Other Resources

COMTIContext Keywords

LibNameOverride

When using the OS/400 DPC programming model, LibNameOverride identifies the library where the OS/400 operating system should look to locate the executable program.

See Also

Other Resources

COMTIContext Keywords

OverrideSourceTP

When using the CICS LU6.2 Link programming model, OverrideSourceTP contains an identifier that reflects the originating transaction (TI application). In CICS, this identifier is typically used for securing CICS resource access for DB2 and VSAM.

See Also

Other Resources

COMTIContext Keywords

PASSWORD

Use the **PASSWORD** keyword, combined with **USERID**, to provide explicit security without the need for a callback. The COMTIContext context name is **PASSWORD**; the value must be a string that contains a valid mainframe security password.

See Also

Tasks

How to Pass a Custom TRM

Reference

REOverride

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PortOverride

When using a TCP/IP transport, PortOverride contains the TCP/IP port number that the Transaction Integrator runtime will use instead of the predefined port in the Remote Environment definition

See Also

Other Resources

COMTIContext Keywords

ProgNameOverride

Use the **ProgNameOverride** keyword to override the program name assigned within the type library and to specify a different program name to be sent to the host. The COMTIContext context name is **ProgNameOverride**; the value must be a string that contains a valid mainframe program name.

See Also

Tasks

How to Pass a Custom TRM

Reference

REOverride

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

PASSWORD

RecvTimeOut

RecvTimeOut indicates the number of seconds that the Transaction Integrator runtime will wait for a response from the host. RecTimeOut is used in place of the predefined RecvTimeOut value in the Remote Environment Definition.

See Also

Other Resources

COMTIContext Keywords

REOverride

Use the **REOverride** keyword to override the type library assignment of the method call to a remote environment (RE) and to assign the call to a newly specified RE. The COMTIContext context name is **REOverride**; the value must be a valid remote environment name.

See Also

Tasks

How to Pass a Custom TRM

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

SendTimeOut

SendTimeOut indicates the number of seconds that the Transaction Integrator runtime will wait for completion of a send operation to the host. This value is used in place of the predefined port in the Remote Environment definition.

See Also

Other Resources

COMTIContext Keywords

TPNameOverride

Use the **TPNameOverride** keyword to override the transaction ID assigned within the type library and to specify the new transaction ID to be sent to the host. The COMTIContext context name is **TPNameOverride**; the value must be a string that contains a valid mainframe transaction ID.

See Also

Tasks

How to Pass a Custom TRM

Reference

REOverride

Concepts

USERID

IMS_LTERM

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

TRMIN

Use the **TRMIN** keyword to override the default transaction request message (TRM) containing the transaction program ID, user ID, password, and other administrative data sent to the host. The COMTIContext context name is **TRMIN**. The TRM must be defined as a user-defined type (UDT), and the name of that UDT must begin with the characters TRMIN.

See Also

Tasks

How to Pass a Custom TRM

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

TRMOUT

Use the **TRMOUT** keyword to override the default transaction request message (TRM) containing the transaction program ID, user ID, password, and other administrative data sent from the host. The COMTIContext context name is **TRMOUT**. The TRM must be defined as a user-defined type (UDT), and the name of that UDT must begin with the characters TRMOUT.

See Also

Tasks

How to Pass a Custom TRM

Concepts

TPNameOverride

USERID

IMS_LTERM

TRMIN

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

USERID

Use the **USERID** keyword, combined with **PASSWORD**, to provide explicit security without the need for a callback. The COMTIContext context name is **USERID**; the value must be a string that contains a valid mainframe security user ID.

See Also

Tasks

How to Pass a Custom TRM

Reference

REOverride

Concepts

TPNameOverride

IMS_LTERM

TRMIN

TRMOUT

Using Custom TRMs and ELMs with COMTIContext

Other Resources

ProgNameOverride

PASSWORD

TI Component Properties

In the Microsoft® Visual Studio® .NET 2003 environment, Transaction Integrator (TI) component properties appear and are edited in the **Properties** pane. When you select an entity in the Visual Studio tree view or list view, its properties appear in categories in the **Properties** pane. When you select multiple items, the intersection of their properties appear to allow for bulk edit.

Note
The properties of the TI components are not intended to be set or changed programmatically. Setting or changing the properties programmatically may cause the component to function incorrectly.

See Also

Other Resources

Properties (TI Project)

Standard Transaction Request and Reply Messages

To support the TCP transaction request message (TRM) Link programming model and the TCP TRM User Data programming model, Transaction Integrator (TI) supports two variations of the CICS Standard Listener TRM.

In This Section

TRM Format for the TCP TRM Link Programming Model

This topic describes the format and content of the transaction request message (TRM) used by the TCP TRM Link programming model.

TRM Request Message

The following table shows the contents of the request message.

TranID	Comma	Client in data
4	1	35

TranID

Transaction ID of the Concurrent Server to be started by the Listener.

Comma

A comma (,) separates the transaction ID from the Client in data.

Client in data

35 bytes of data used by the CICS TCP/IP security exit and passed to the Concurrent Server in the transaction initiation message (TIM).

Client in data for Microsoft Security Exit format

The following code block describes the format of the client in data for the Microsoft security exit.

Syntax

```
struct CLIENT_IN_DATA {
   BYTE    bUserID[8];
   BYTE    bPassword[8];
   BYTE    bLinkToName[8];
   USHORT  usCommareaLen;
   BYTE    bReserved[9];
} UNALIGNED;
```

Client in data for IBM Security Exit format

The following code block describes the format of the client in data for the IBM security exit.

Syntax

```
struct CLIENT_IN_DATA2 {
   BYTE    bSecFlag;
   BYTE    bPassword[8];
   BYTE    bUserID[8];
   BYTE    bLinkToName[8];
   USHORT  usCommareaLen;
   BYTE    bReserved[8];
} UNALIGNED;
```

TRM Reply Message

The following table shows the contents of the reply message.

TRM reply msg length	Formatted field length	Formatted field code	*Data*
4	4	1	*0-n*

Note
The formatted field length, formatted field code, and data can be repeated multiple times in a single message.

TRM reply msg length

The total length of the TRM reply message. This length is the sum of all the lengths of the formatted fields that follow in the message and does not include the length of the TRM reply msg length field itself.

Formatted field length

The length of the formatted field.

The formatted field length is the sum of the combination of the Formatted field code length and the Data length.

Formatted field code

A 1-byte code that describes the information passed from the Concurrent Server back to the client.

You cannot change the Formatted field code.

The field codes are specific to the communication handling between the WIP and HIP TCP Transports and the MSCMTICS, MSHIPLNK and TCP Concurrent Server programs.

Data

0 or more bytes of information that is associated with a specific formatted field.

You may change the information stored in Data. If you change Data, be sure that you also change the TRM Reply and the Formatted Field Length to the new values.

The length of Data is equal to the formatted field length minus the size of the formatted field code.

Normal codes

The following table shows the meaning of the normal codes.

Code	Type	Meaning
0x01	Info	Version ID for Microsoft® Transaction Integrator Concurrent Server
0x02	Info	User Data
0x07	Info	Execution OK

Error codes

The following table shows the meaning of the error codes.

Code	Type	Meaning
0x03	Error	Invalid ProgID
0x04	Error	Invalid TranID
0x05	Error	Inquiry Failed
0x06	Error	Inquiry Status
0x08	Error	Program ABEND
0x09	Error	Execution Failed
0x0A	Error	Invalid TRM

For more information about the format of the TRM, see the TRM definition file at <drive>:\Program Files\Microsoft Host Integration Server\System\TIM\MicrosoftTRMDefs.tim. Use Microsoft Visual Studio® .NET 2003 to view the file.

See Also

Reference

TRM Format for the TCP TRM User Data Programming Model

TRM Format for the TCP TRM User Data Programming Model

This section describes the format and content of the transaction request message (TRM) used by the TCP TRM User Data programming model.

TRM Request Message

The following table shows the contents of the request message.

TranID	Comma	Client in data
4	1	35

TranID

Transaction ID of the Concurrent Server to be started by the Listener.

Comma

A comma (,) separates the transaction ID from the Client in data.

Client in data

35 bytes of data used by the CICS TCP/IP security exit and passed to the Concurrent Server in the transaction initiation message (TIM).

Client in data for Microsoft Security Exit format

The following code block describes the format of the client in data for the Microsoft security exit.

```
struct CLIENT_IN_DATA {
   BYTE    bUserID[8];
   BYTE    bPassword[8];
   BYTE    bReserved[19];
} UNALIGNED;
```

Client in data for IBM Security Exit format

The following code block describes the format of the client in data for the IBM security exit.

```
struct CLIENT_IN_DATA2 {
   BYTE    bSecFlag;
   BYTE    bPassword[8];
   BYTE    bUserID[8];
   BYTE    bReserved[18];
} UNALIGNED;
```

Client in data for COBOL

The following code block describes the format of the client in COBOL

```
01 CLIENT-IN-DATA                      PIC X(35).
       01 FILLER REDEFINES CLIENT-IN-DATA.
          05 CID-USERID                PIC X(8).
          05 CID-PASSWORD              PIC X(8).
          05 CID-LINK-TO-PROG          PIC X(8).
          05 CID-COMMAREA-LEN          PIC S9(4) COMP.
          05 CID-DATA-LEN              PIC S9(8) COMP.
          05 CID-VERSION               PIC X.
          05 CID-FLAG-1                PIC X.
          05 CID-FLAG-2                PIC X.
          05 CID-RESERVED              PIC X.
          05 CID-FORMAT                PIC X.
```

Client in data Constants for COBOL

The following code block describes the constants for the client in data in COBOL.

```
01 CLIENT-IN-DATA-CONSTANTS.
05 CID-C-VERSION.
10 CID-VERSION-1 PIC X VALUE X'00'.
10 CID-VERSION-2 PIC X VALUE X'01'.
05 CID-C-FLAG-1.
10 CID-USE-TICS-WORK-AREA PIC X VALUE X'01'.
05 CID-C-FLAG-2.
10 CID-PC-NONE PIC X VALUE X'01'.
10 CID-PC-OPEN PIC X VALUE X'02'.
10 CID-PC-USE PIC X VALUE X'04'.
10 CID-PC-CLOSE PIC X VALUE X'08'.
10 CID-NO-OBJ-PERSIST PIC X VALUE X'10'.
05 CID-C-FORMAT.
10 CID-FORMAT-NOTSET PIC X VALUE X'00'.
10 CID-FORMAT-MS PIC X VALUE X'01'.
10 CID-FORMAT-IBM PIC X VALUE X'02'.
```

TRM Reply Message

The following table shows the contents of the reply message.

TRM reply msg length	**Formatted field length**	**Formatted field code**	***Data***
2	4	1	*0-n*

Note
The formatted field length, formatted field code, and data can be repeated multiple times in a single message.

TRM reply msg length

The total length of the TRM reply message. This length is the sum of all the lengths of the formatted fields that follow in the message and does not include the length of the TRM reply msg length field itself.

Formatted field length

The length of the formatted field.

The formatted field length is the sum of the combination of the Formatted field code length and the Data length.

Formatted field code

A 1-byte code that describes the information passed from the Concurrent Server back to the client.

You cannot change the Formatted field code.

The field codes are specific to the communication handling between the WIP and HIP TCP Transports and the MSCMTICS, MSHIPLNK and TCP Concurrent Server programs.

Data

0 or more bytes of information that is associated with a specific formatted field.

You may change the information stored in Data. If you change Data, be sure that you also change the TRM Reply and the Formatted Field Length to the new values.

The length of Data is equal to the formatted field length minus the size of the formatted field code.

Normal codes

The following table shows the meaning of the normal codes.

Code	**Type**	**Meaning**
0x01	Info	Version ID for Microsoft® Transaction Integrator Concurrent Server

0x02	Info	User Data
0x07	Info	Execution OK

Error codes

The following table shows the meaning of the error codes.

Code	Type	Meaning
0x03	Error	Invalid ProgID
0x04	Error	Invalid TranID
0x05	Error	Inquiry Failed
0x06	Error	Inquiry Status
0x08	Error	Program ABEND
0x09	Error	Execution Failed
0x0A	Error	Invalid TRM
0x0B	Error	Server generated an exception
0x0C	Error	Exception error information is in the Meta Data Error Block

For more information about the format of the TRM, see the TRM definition file at <drive>:\Program Files\ Microsoft Host IntegrationServer\System\TIM\MicrosoftTRMDefs.tim. Use Microsoft Visual Studio® .NET 2003 to view the file.

See Also

Reference

TRM Format for the TCP TRM Link Programming Model

CICS Enhanced Listener Request and Reply Messages

To support the CICS TCP enhanced listener message (ELM) Link programming model and the TCP ELM User Data programming model, Transaction Integrator (TI) supports two variations of the CICS Enhanced Listener TRM.

In This Section

ELM Format for the TCP ELM Link Programming Model

This section describes the format and content of the enhanced listener message (ELM) used by the TCP ELM Link programming model.

ELM Request Message

The following table shows the contents of the request message.

Client in data
35

Client in data

35 bytes of data used by the CICS TCP/IP security exit and passed to the Concurrent Server in the Transaction Integrator metadata (TIM) file.

Client in data for Microsoft Security Exit format

The following code block describes the format of the client in data for the Microsoft security exit.

```
struct CLIENT_IN_DATA {
   BYTE    bUserID[8];
   BYTE    bPassword[8];
   BYTE    bLinkToName[8];
   USHORT  usCommareaLen;
   BYTE    bReserved[9];
} UNALIGNED;
```

Client in data for IBM Security Exit format

The following code block describes the format of the client in data for the IBM security exit.

```
struct CLIENT_IN_DATA2 {
   BYTE    bSecFlag;
   BYTE    bPassword[8];
   BYTE    bUserID[8];
   BYTE    bLinkToName[8];
   USHORT  usCommareaLen;
   BYTE    bReserved[8];
} UNALIGNED;
```

Client in data for COBOL

The following code block describes the format of the client in COBOL

```
01 CLIENT-IN-DATA                      PIC X(35).
       01 FILLER REDEFINES CLIENT-IN-DATA.
          05 CID-USERID                PIC X(8).
          05 CID-PASSWORD              PIC X(8).
          05 CID-LINK-TO-PROG          PIC X(8).
          05 CID-COMMAREA-LEN          PIC S9(4) COMP.
          05 CID-DATA-LEN              PIC S9(8) COMP.
          05 CID-VERSION               PIC X.
          05 CID-FLAG-1                PIC X.
          05 CID-FLAG-2                PIC X.
          05 CID-RESERVED              PIC X.
          05 CID-FORMAT                PIC X.
```

Client in data Constants for COBOL

The following code block describes the constants for the client in data in COBOL.

```
01 CLIENT-IN-DATA-CONSTANTS.

05 CID-C-VERSION.
```

```
10 CID-VERSION-1 PIC X VALUE X'00'.
10 CID-VERSION-2 PIC X VALUE X'01'.
05 CID-C-FLAG-1.
10 CID-USE-TICS-WORK-AREA PIC X VALUE X'01'.
05 CID-C-FLAG-2.
10 CID-PC-NONE PIC X VALUE X'01'.
10 CID-PC-OPEN PIC X VALUE X'02'.
10 CID-PC-USE PIC X VALUE X'04'.
10 CID-PC-CLOSE PIC X VALUE X'08'.
10 CID-NO-OBJ-PERSIST PIC X VALUE X'10'.
05 CID-C-FORMAT.
10 CID-FORMAT-NOTSET PIC X VALUE X'00'.
10 CID-FORMAT-MS PIC X VALUE X'01'.
10 CID-FORMAT-IBM PIC X VALUE X'02'.
```

ELM Reply Message

The following table shows the contents of the reply message.

ELM reply msg length	Formatted field length	Formatted field code	*Data*
4	4	1	*0-n*

Note
The formatted field length, formatted field code, and data can be repeated multiple times in a single message.

ELM reply msg length

The total length of the ELM reply message. This length is the sum of all the lengths of the formatted fields that follow in the message and does not include the length of the ELM reply msg length field itself.

Formatted field length

The length of the formatted field.

The formatted field length is the sum of the combination of the Formatted field code length and the Data length.

Formatted field code

A 1-byte code that describes the information passed from the Concurrent Server back to the client.

You cannot change the Formatted field code.

The field codes are specific to the communication handling between the WIP and HIP TCP Transports and the MSCMTICS, MSHIPLNK and TCP Concurrent Server programs.

Data

0 or more bytes of information that is associated with a specific formatted field.

You may change the information stored in Data. If you change Data, be sure that you also change the TRM Reply and the Formatted Field Length to the new values.

The length of Data is equal to the formatted field length minus the size of the formatted field code.

Normal codes

The following table shows the meaning of the normal codes.

Code	Type	Meaning
0x01	Info	Version ID for Microsoft® Transaction Integrator Concurrent Server
0x02	Info	User Data

0x07	Info	Execution OK

Error codes

The following table shows the meaning of the error codes.

Code	Type	Meaning
0x03	Error	Invalid ProgID
0x04	Error	Invalid TranID
0x05	Error	Inquiry Failed
0x06	Error	Inquiry Status
0x08	Error	Program ABEND
0x09	Error	Execution Failed
0x0A	Error	Invalid ELM

For more information about the format of the TRM, see the TRM definition file at <drive>:\Program Files\ Microsoft Host Integration Server\System\TIM\MicrosoftTRMDefs.tim. Use Microsoft Visual Studio® .NET 2003 to view the file.

See Also

Reference

ELM Format for the TCP ELM User Data Programming Model

Other Resources

Enhanced Listener CICS Administration

ELM Format for the TCP ELM User Data Programming Model

This section describes the format and content of the enhanced listener message (ELM) used by the TCP ELM User Data programming model.

ELM Request Message

The following table shows the contents of the request message.

Client in data
35

Client in data

35 bytes of data used by the CICS TCP/IP security exit and passed to the Concurrent Server in the transaction initiation message (TIM).

Client in data for Microsoft Security Exit format

The following code block describes the format of the client in data for the Microsoft security exit.

Syntax

```
struct CLIENT_IN_DATA {
   BYTE    bUserID[8];
   BYTE    bPassword[8];
   BYTE    bReserved[19];
} UNALIGNED;
```

Client in data for IBM Security Exit format

The following code block describes the format of the client in data for the IBM security exit.

Syntax

```
struct CLIENT_IN_DATA2 {
   BYTE    bSecFlag;
   BYTE    bPassword[8];
   BYTE    bUserID[8];
   BYTE    bReserved[18];
} UNALIGNED;
```

ELM Reply Message

The following table shows the contents of the reply message.

ELM reply msg length	Formatted field length	Formatted field code	*Data*
4	4	1	*0-n*

Note
The formatted field length, formatted field code, and data can be repeated multiple times in a single message.

ELM reply msg length

The total length of the ELM reply message. This length is the sum of all the lengths of the formatted fields that follow in the message and does not include the length of the ELM reply msg length field itself.

Formatted field length

The length of the formatted field.

The formatted field length is the sum of the combination of the Formatted field code length and the Data length.

Formatted field code

A 1-byte code that describes the information passed from the Concurrent Server back to the client.

You cannot change the Formatted field code.

The field codes are specific to the communication handling between the WIP and HIP TCP Transports and the MSCMTICS, MSHIPLNK and TCP Concurrent Server programs.

Data

0 or more bytes of information that is associated with a specific formatted field.

You may change the information stored in Data. If you change Data, be sure that you also change the TRM Reply and the Formatted Field Length to the new values.

The length of Data is equal to the formatted field length minus the size of the formatted field code.

Normal codes

The following table shows the meaning of the normal codes.

Code	Type	Meaning
0x01	Info	Version ID for Microsoft Transaction Integrator Concurrent Server
0x02	Info	User Data
0x07	Info	Execution OK

Error codes

The following table shows the meaning of the error codes.

Code	Type	Meaning
0x03	Error	Invalid ProgID
0x04	Error	Invalid TranID
0x05	Error	Inquiry Failed
0x06	Error	Inquiry Status
0x08	Error	Program ABEND
0x09	Error	Execution Failed
0x0A	Error	Invalid ELM

For more information about the format of the TRM, see the TRM definition file at <drive>:\Program Files\ Microsoft Host Integration Server\System\TIM\MicrosoftTRMDefs.tim. Use Microsoft Visual Studio® .NET 2003 to view the file.

See Also

Reference

ELM Format for the TCP ELM Link Programming Model

Other Resources

Enhanced Listener CICS Administration

Enhanced Listener CICS Administration

The following code defines a CICS Enhanced Listener. There are several new keywords available for use with the Enhanced Listener. The parameter definitions describe how these new Listener configuration values are used for support the TI Enhanced Listener feature.

```
      EZACICD TYPE=LISTENER,  Listener record definition              X
             FORMAT=ENHANCED,  Enhanced Listener                      X
             APPLID=XYZ12345,  Application ID of CICS region          X
             TRANID=CSKM,      Transaction name for Listener          X
             PORT=1234,        Port number for Listener               X
             IMMED=YES,        Listener starts up at initialization?  X
             NUMSOCK=50,       Number of sockets supported by Listener X
             ACCTIME=30,       Timeout value for Accept               X
             GIVTIME=30,       Timeout value for Givesocket           X
             REATIME=30,       Timeout value for Read                 X
             CSTRAN=MSCS,      Name of child server transaction       X
             CSSTTYPE=KC,       Child server startup type             X
             CSDELAY=000000,   Child server delay interval            X
             MSGLEN=35,        Length of input message                X
             PEEKDATA=NO,       Peek option                           X
             MSGFORM=EBCDIC,   Output message format                  X
```

CSTRANID

This parameter is specific to the enhanced version of the Listener and specifies the default child server transaction that the Listener starts.

For enhanced listener message (ELM) Link support, this value should be set to MSCS to conform to the samples that Microsoft delivers with the product. The MSCS transaction code should be associated with the mscmtics.cbl program that supports the Standard and Enhanced Listener protocols. Otherwise, this parameter is the transaction ID that will be executed for each request made to the designated port.

CSSTTYPE

This parameter is specific to the enhanced version of the Listener and specifies the default start method for the child server task. This parameter can be overridden by the security/transaction exit. Possible values are IC, KC, and TD.

IC

Indicates that the child server task is started using EXEC CICS START with the value specified by CSDLYINT (or an overriding value from the security/transaction exit) as the delay interval.

KC

Indicates that the child server task is started using EXEC CICS START with no delay interval.

TD

Indicates that the child server task is started using the EXEC CICS WRITEQ TD command, which uses transient data to trigger the child server task.

CSDLYINT

This parameter is specific to the enhanced version of the Listener and is applicable only if CSSTTYPE is IC. It specifies the delay interval to be used on the EXEC CICS START command, in the form hhmmss (hours/minutes/seconds).

MSGFORM

This parameter is specific to the enhanced version of the Listener and indicates whether an error message returned to the client should be in ASCII or Extended Binary Coded Decimal Interchange Code (EBCDIC) format. ASCII is the default. MSGFORM is displayed as MSGFORMat on the IBM-supplied CICS Transaction screens.

For TI Enhanced Listener support, this value must be set to EBCDIC.

MSGLEN

This parameter is specific to the enhanced version of the Listener and specifies the length of the data to be received from the client by the Listener. The valid range is from 0 through 999. If the value is 0, the Listener does not read in any data from the client.

For TI Enhanced Listener support, this value must be the size of the ELM that is delivered. The size of the ELM is 35.

PEEKDATA

This parameter is specific to the enhanced version of the Listener and applies only if MSGLEN is not 0.

A value of NO indicates that the Listener performs a normal read of the client data. The child server application accesses this data in the data area-2 portion of the transaction initiation message (TIM).

A value of YES indicates that the Listener reads the data using the Peek option. The data remains queued in TCP/IP and the child server applications read it in rather than accessing it through the TIM.

For TI Enhanced Listener support, this value must be set to NO. Setting this value to NO instructs the Enhanced Listener to read the ELM (35 bytes) and place it in the TIM in the data area-2 field.

The mscmtics.cbl Concurrent Server uses the information in this area to determine what server program to link to.

For more information about the format of the ELM, see the ELM definition file at <drive>:\Program Files\Microsoft Host Integration Server\System\TIM\MicrosoftELMDefs.tim. Use Microsoft Visual Studio® .NET 2003 to view the file.

See Also
Reference
ELM Format for the TCP ELM Link Programming Model
ELM Format for the TCP ELM User Data Programming Model

Microsoft Concurrent Server

The MSCS transaction (program mscmtics.cbl) samples support both the Standard and the Enhanced Listener. The transaction program can be started by either the Enhanced or Standard Listener.

Each listener passes a unique transaction initiation message (TIM) to the transaction program when the Concurrent Server is started. The Standard Listener formats and passes the TIM shown in the following code sample. The length of this TIM is 72 bytes.

```
01  TRANSACTION-INITIATION-MESSAGE.
    05  GIVE-TAKE-SOCKET    PIC 9(8) COMP.
    05  LSTN-NAME           PIC X(8).
    05  LSTN-SUBNAME        PIC X(8).
    05  CLIENT-IN-DATA      PIC X(35).
    05  FILLER              PIC X(1).
    05  SOCKADDR-IN-PARM.
        15 SIN-FAMILY       PIC 9(4) COMP.
        15 SIN-PORT         PIC 9(4) COMP.
        15 SIN-ADDRESS      PIC 9(8) COMP.
        15 SIN-ZERO         PIC X(8).
```

The Enhanced Listener formats and passes the TIM shown in the following code sample. The length of this TIM is 189 bytes.

```
01  TRANSACTION-INITIATION-MESSAGE.
    05  GIVE-TAKE-SOCKET    PIC 9(8) COMP.
    05  LSTN-NAME           PIC X(8).
    05  LSTN-SUBNAME        PIC X(8).
    05  CLIENT-IN-DATA      PIC X(35).
    05  FILLER              PIC X(1).
    05  SOCKADDR-IN-PARM.
        15 SIN-FAMILY       PIC 9(4) COMP.
        15 SIN-PORT         PIC 9(4) COMP.
        15 SIN-ADDRESS      PIC 9(8) COMP.
        15 SIN-ZERO         PIC X(8).
    05  FILLER              PIC X(80).
    05  DATA-AREA-2-LEN     PIC 9(4) COMP.
    05  DATA-AREA-2         PIC X(35).
```

The mscmtics.cbl sample Concurrent Server can determine whether the Standard or the Enhanced Listener was used by evaluating the length of the TIM received.

In a scenario where the Enhanced Listener started the Microsoft Concurrent Server, the mscmtics.cbl program looks at the Client-in-data that is contained in the ELM found in the TIM data area-2 field. The Client-in-data contains the name of the CICS Server program to be executed and the length of the request data to be received from the client. The following code sample shows the contents of this data area.

```
       01 CLIENT-IN-DATA                   PIC X(35).
       01 FILLER REDEFINES CLIENT-IN-DATA.
          05 CID-USERID                    PIC X(8).
          05 CID-PASSWORD                  PIC X(8).
          05 CID-LINK-TO-PROG              PIC X(8).
          05 CID-COMMAREA-LEN              PIC S9(4) COMP.
          05 CID-DATA-LEN                  PIC S9(8) COMP.
          05 CID-VERSION                   PIC X.
             88 CID-VERSION-1              VALUE X'00'.
             88 CID-VERSION-2              VALUE X'01'.
          05 CID-FLAGS                     PIC X(2).
             88 CID-FLAGS-PERSISTENT-NONE  VALUE X'0001'.
             88 CID-FLAGS-PERSISTENT-OPEN  VALUE X'0002'.
             88 CID-FLAGS-PERSISTENT-USE   VALUE X'0004'.
             88 CID-FLAGS-PERSISTENT-CLOSE VALUE X'0008'.
          05 CID-RESERVED                  PIC X.
          05 CID-FORMAT                    PIC X.
```

```
           88 CID-FORMAT-NOTSET           VALUE X'00'.
           88 CID-FORMAT-MS                VALUE X'01'.
           88 CID-FORMAT-IBM               VALUE X'02'.
```

See Also

Other Resources

Standard Transaction Request and Reply Messages

CICS Enhanced Listener Request and Reply Messages

Data Integration Programmer's Reference

This section of the Host Integration Server 2009 Programmer's Guide describes the objects, methods, properties, controls, and other interfaces that enable you to integrate data into your Host Integration Server application.

For general information about programming for data integration, see Data Integration Programmer's Guide.

For sample code using data integration, see Data Integration Samples section of the SDK.

In This Section

OLE DB Providers Programmer's Reference

ODBC Driver for DB2 Programmer's Reference

Managed Provider for DB2 Programmer's Reference

Data Access Library Programmer's Reference

Managed Data Provider for Host Files Programmer's Reference

ActiveX Controls Programmer's Reference

ADO Programmer's Reference

OLE DB Providers Programmer's Reference

The OLE DB specification version 2.0 defines a number of objects and interfaces.

Microsoft OLE DB Provider for AS/400 and VSAM supports the OLE DB objects and interfaces appropriate for an OLE DB data provider accessing a non-SQL host file system.

Microsoft OLE DB Provider for DB2 supports the OLE DB objects and interfaces appropriate for an OLE DB data provider accessing an SQL database.

This section also provides a comparison of the objects and interfaces supported by OLE DB Provider for AS/400 and VSAM and OLE DB Provider for DB2.

Note
Do not use double or real columns when specifying **UPDATE** or **DELETE** statements on the MsDb2DataAdapter. These rows will not be found and will throw the concurrency violation exception.

In This Section

OLE DB Object Support Comparison

The following table compares the OLE DB version 2.0 objects that are supported by the current version of Microsoft® OLE DB Provider for AS/400 and VSAM and Microsoft OLE DB Provider for DB2.

OLE DB object	OLE DB Provider for AS/400 and VSAM	OLE DB Provider for DB2
Command	Yes, most interfaces	Yes, most interfaces
CustomErrorObject	No	Yes, all interfaces
DataSource	Yes, most interfaces	Yes, some interfaces
Enumerator	No	No
ErrorObject	Yes, all interfaces	Yes, all interfaces
ErrorRecord	Yes, all interfaces	Yes, all interfaces
Index	Yes, all interfaces	No
MultipleResults	No	No
Rowset	Yes, most interfaces	Yes, some interfaces
Session	Yes, some interfaces	Yes, some interfaces
Transaction	No	Yes, some interfaces
TransactionOptions	No	Yes, all interfaces
View	Yes, all interfaces	No

OLE DB Interface Support Comparison

The following table compares the OLE DB version 2.0 interfaces that are supported by the current version of Microsoft® OLE DB Provider for AS/400 and VSAM and Microsoft OLE DB Provider for DB2.

Object	Interface	OLE DB Provider for AS/400 and VSAM	OLE DB Provider for DB2
Command	**IAccessor**	Yes	Yes
	IColumnsInfo	Yes	Yes
	IColumnsRowset	No	No
	ICommand	Yes	Yes
	ICommandPersist	No	No
	ICommandPrepare	No	Yes
	ICommandProperties	Yes	Yes
	ICommandText	Yes	Yes
	ICommandWithParameters	No	Yes
	IConvertType	Yes	Yes
	ISupportErrorInfo	Yes	Yes
CustomErrorObject	**IErrorLookup**	No	Yes
	ISQLErrorInfo	No	Yes
DataSource	**IDBAsynchStatus**	No	No
	IConnectionPointContainer	No	No
	IDBCreateSession	Yes	Yes
	IDBDataSourceAdmin	No	No
	IDBInfo	No	Yes
	IDBInitialize	Yes	Yes
	IDBProperties	Yes	Yes
	IPersist	Yes	No
	IPersistFile	Yes	No
	ISupportErrorInfo	Yes	Yes
Enumerator	**IDBInitialize**	No	No

	IDBProperties	No	No
	IParseDisplayName	No	No
	ISourcesRowset	No	No
	ISupportErrorInfo	No	No
ErrorObject	**IErrorRecords**	Yes	Yes
ErrorRecord	**IErrorInfo**	Yes	Yes
Index	**IAccessor**	Yes	No
	IColumnsInfo	Yes	No
	IConvertType	Yes	No
	IRowset	Yes	No
	IRowsetChange	Yes	No
	IRowsetFind	Yes	No
	IRowsetIdentity	Yes	No
	IRowsetIndex	Yes	No
	IRowsetInfo	Yes	No
	IRowsetLocate	Yes	No
	IRowsetRefresh	Yes	No
	IRowsetScroll	Yes	No
	IRowsetUpdate	Yes	No
	IRowsetView	Yes	No
	ISupportErrorInfo	Yes	No
MultipleResults	**IMultipleResults**	No	No
	ISupportErrorInfo	No	No
Rowset	**IAccessor**	Yes	Yes
	IChapteredRowset	Yes	No
	IColumnsInfo	Yes	Yes
	IColumnsRowset	Yes	No

	IConnectionPointContainer	No	No
	IConvertType	Yes	Yes
	IDBAsynchStatus	No	No
	IRowset	Yes	Yes
	IRowsetChange	Yes	Yes
	IRowsetChapterMember	No	No
	IRowsetFind	Yes	No
	IRowsetIdentity	Yes	No
	IRowsetIndex	Yes	No
	IRowsetInfo	Yes	Yes
	IRowsetLocate	Yes	No
	IRowsetRefresh	Yes	No
	IRowsetScroll	No	No
	IRowsetUpdate	Yes	Yes
	IRowsetView	Yes	No
	ISupportErrorInfo	Yes	Yes
Session	**IAlterIndex**	No	No
	IAlterTable	No	No
	IDBCreateCommand	Yes	Yes
	IDBSchemaRowset	Yes	Yes
	IGetDataSource	Yes	Yes
	IIndexDefinition	No	No
	IOpenRowset	Yes	Yes
	ISessionProperties	Yes	Yes
	ISupportErrorInfo	Yes	Yes
	ITableDefinition	No	No

	ITransaction	No	Yes
	ITransactionJoin	No	No
	ITransactionLocal	No	Yes
	ITransactionObject	No	Yes
Transaction	**IConnectionPointContainer**	No	No
	ISupportErrorInfo	No	No
	ITransaction	No	No
TransactionOptions	**ISupportErrorInfo**	No	Yes
	ITransactionOptions	No	Yes
View	**IAccessor**	Yes	No
	IColumnsInfo	Yes	No
	ISupportErrorInfo	Yes	No
	IViewChapter	Yes	No
	IViewFilter	Yes	No
	IViewRowset	Yes	No
	IViewSort	Yes	No

OLE DB Object Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the OLE DB version 2.0 objects that are supported by the current version of Microsoft® OLE DB Provider for AS/400 and VSAM.

OLE DB object	Support
Command Object	Yes, most interfaces
CustomErrorObject	No
DataSource Object	Yes, most interfaces
Enumerator	No
ErrorObject Object	Yes, all interfaces
ErrorRecord Object	Yes, all interfaces
Index Object	Yes, all interfaces
MultipleResults	No
Rowset Object	Yes, most interfaces
Session Object	Yes, some interfaces
Transaction	No
TransactionOptions	No
View Object	Yes, all interfaces

In This Section

- OLE DB Interface Support in the OLE DB Provider for AS/400 and VSAM
- Command Object (OLE DB Provider for AS/400 and VSAM)
- DataSource Object (OLE DB Provider for AS/400 and VSAM)
- ErrorObject Object (OLE DB Provider for AS/400 and VSAM)
- ErrorRecord Object (OLE DB Provider for AS/400 and VSAM)
- Index Object (OLE DB Provider for AS/400 and VSAM)
- Rowset Object (OLE DB Provider for AS/400 and VSAM)
- Session Object (OLE DB Provider for AS/400 and VSAM)
- View Object (OLE DB Provider for AS/400 and VSAM)
- OLE DB Property Support in the OLE DB Provider for AS/400 and VSAM

OLE DB Interface Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the OLE DB version 2.0 interfaces that are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM.

Object	Interface	Support
Command Object	**IAccessor**	Yes
	IColumnsInfo	Yes
	IColumnsRowset	No
	ICommand	Yes
	ICommandPersist	No
	ICommandPrepare	No
	ICommandProperties	Yes
	ICommandText	Yes
	ICommandWithParameters	No
	IConvertType	Yes
	ISupportErrorInfo	Yes
CustomErrorObject	**IErrorLookup**	No
	ISQLErrorInfo	No
DataSource Object	**IDBAsynchStatus**	No
	IDBConnectionPointContainer	No
	IDBCreateSession	Yes
	IDBDataSourceAdmin	No
	IDBInfo	No
	IDBInitialize	Yes
	IDBProperties	Yes
	IPersist	Yes
	IPersistFile	Yes
	ISupportErrorInfo	Yes

Enumerator	**IDBInitialize**	No
	IDBProperties	No
	IParseDisplayName	No
	ISourcesRowset	No
	ISupportErrorInfo	No
ErrorObject Object	**IErrorRecords**	Yes
ErrorRecord Object	**IErrorInfo**	Yes
Index Object	**IAccessor**	Yes
	IColumnsInfo	Yes
	IConvertType	Yes
	IRowset	Yes
	IRowsetChange	Yes
	IRowsetFind	Yes
	IRowsetIdentity	Yes
	IRowsetIndex	Yes
	IRowsetInfo	Yes
	IRowsetLocate	Yes
	IRowsetRefresh	Yes
	IRowsetScroll	Yes
	IRowsetUpdate	Yes
	IRowsetView	Yes
	ISupportErrorInfo	Yes
MultipleResults	**IMultipleResults**	No
	ISupportErrorInfo	No
Rowset Object	**IAccessor**	Yes
	IChapteredRowset	Yes
	IColumnsInfo	Yes

	IColumnsRowset	No
	IConnectionPointContainer	No
	IConvertType	Yes
	IDBAsynchStatus	No
	IRowset	Yes
	IRowsetChange	Yes
	IRowsetChapterMember	No
	IRowsetFind	Yes
	IRowsetIdentity	Yes
	IRowsetIndex	Yes
	IRowsetInfo	Yes
	IRowsetLocate	Yes
	IRowsetRefresh	Yes
	IRowsetScroll	No
	IRowsetUpdate	Yes
	IRowsetView	Yes
	ISupportErrorInfo	Yes
Session Object	**IAlterIndex**	No
	IAlterTable	No
	IDBCreateCommand	Yes
	IDBSchemaRowset	Yes
	IGetDataSource	Yes
	IIndexDefinition	No
	IOpenRowset	Yes
	ISessionProperties	Yes
	ISupportErrorInfo	Yes

	ITableDefinition	No
	ITransaction	No
	ITransactionJoin	No
	ITransactionLocal	No
	ITransactionObject	No
Transaction	**IConnectionPointContainer**	No
	ISupportErrorInfo	No
	ITransaction	No
TransactionOptions	**ISupportErrorInfo**	No
	ITransactionOptions	No
View Object	**IAccessor**	Yes
	IColumnsInfo	Yes
	ISupportErrorInfo	Yes
	IViewChapter	Yes
	IViewFilter	Yes
	IViewRowset	Yes
	IViewSort	Yes

Command Object (OLE DB Provider for AS/400 and VSAM)

The **Command** object is created by an OLE DB consumer, or by a service provider on behalf of a consumer. A **Command** object is used to execute a distributed data management (DDM)-specific command on a remote DDM server. The **Command** object currently supports executing Command Language commands on AS/400 DDM servers.

It is important not to confuse a command, which is an OLE COM object, and its command text, which is a string. Commands are generally used for data definition, such as creating a table or granting privileges, and data manipulation, such as updating or deleting rows. A special case of data manipulation using the **Command** object is opening a rowset (a table).

Before a consumer can use a command, it must determine if commands are supported. To do this, the consumer calls **QueryInterface** for **IDBCreateCommand** on a session. If this interface is exposed, the provider supports commands. To create a command, the consumer then calls **IDBCreateCommand::CreateCommand** on the session. A single session can be used to create multiple commands.

When the command is first created, it does not contain a command text. The consumer sets the command text with **ICommandText::SetCommandText**. Because the text command syntax is provider-specific, the consumer passes the globally unique identifier (GUID) of the syntax to use. For use with Microsoft OLE DB Provider for AS/400 and VSAM, the GUID is DBGUID_DBSQL. Note that under OLE DB Provider for AS/400 and VSAM, this GUID does not signify that the text command is a superset of ANSI SQL. The level at which the provider supports ANSI SQL is specified by the DBPROP_SQLSUPPORT property. This property is a bitmask specifying the level of support for SQL. OLE DB Provider for AS/400 and VSAM sets this property to DBPROPVAL_SQL_NONE, indicating that SQL is not supported.

The syntax supported by OLE DB Provider for AS/400 and VSAM for command text is as follows:

```
EXEC COMMAND DDMCmd
```

where *DDMCmd* represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands enable you to request functions from the OS/400 operating system. Some examples are the Delete File (DLTF) or Display File Description DSPFFD) commands. These are the same commands that could be issued at the command prompt if you were connected to an AS/400 through a 5250 terminal session. For a detailed list of possible commands, see the OS/400 CL reference for your platform.

The syntax supported by OLE DB Provider for AS/400 and VSAM to open a rowset (table) using command text is as follows:

```
EXEC OPEN FileName
```

where *FileName* represents one of the following host file naming conventions listed in the following table.

Host file type	File naming convention
VSAM Data Sets	DATASETNAME.FILENAME
Partitioned Data Sets	DATASETNAME.FILENAME(MEMBER)
OS/400 Files	LIBRARY/FILE
OS/400 Files	LIBRARY/FILENAME
OS/400 File Members	LIBRARY/FILE(MEMBER)
OS/400 File Members	LIBRARY.FILENAME(MEMBER)

Note that if a member of a library contains a dot in the member name, the member name must be surrounded by double quotes. For example, if the member name is NAMES.DAT, the proper syntax used to open a rowset using command text is as follows:

```
EXEC OPEN LIBRARY/FILE("NAMES.DAT")
```

To execute the command, the consumer calls **ICommand::Execute**. If the command text specifies the command to open a

rowset, (an **EXEC OPEN** command), **Execute** instantiates the rowset and returns an interface pointer to it.

The following interfaces of the **Command** object are supported by the current version of OLE DB Provider for AS/400 and VSAM:

- **IAccessor**
- **IColumnsInfo**
- **ICommand**
- **ICommandProperties**
- **ICommandText**
- **IConvertType**
- **ISupportErrorInfo**

DataSource Object (OLE DB Provider for AS/400 and VSAM)

The **DataSource** object is created by an OLE DB consumer. The **DataSource** object contains the knowledge and ability to connect to an IBM mainframe or AS/400 computer over Advanced Program-to-Program Communications (APPC) and LU 6.2 (through Microsoft Host Integration Server 2009) or over TCP/IP. The **DataSource** object is used to create one or more **Session** objects.

The following interfaces of the **DataSource** object are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM:

- **IDBCreateSession**
- **IDBInitialize**
- **IDBProperties**
- **IPersist**
- **IPersistFile**
- **ISupportErrorInfo**

ErrorObject Object (OLE DB Provider for AS/400 and VSAM)

The **ErrorObject** object is created by any interface on any SNA OLE DB object. The **ErrorObject** object is used to retrieve additional information when an error occurs.

The following interface of the **ErrorObject** object is supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM:

- **IErrorRecords**

The **IErrorRecords** interface returns **ErrorRecord** objects with detailed information about the error that occurred.

ErrorRecord Object (OLE DB Provider for AS/400 and VSAM)

The **ErrorRecord** object is created by calling the **IErrorRecord** interface on the **ErrorObject** object. An **ErrorObject** is created on any interface on any SNA OLE DB object when an error occurs. The **ErrorRecord** object is used to retrieve additional information when an error occurs.

The following interface of the **ErrorRecord** object is supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM:

- **IErrorInfo**

OLE DB interface methods return error information in two ways. The error code returned by an interface method, known as the **return code**, indicates the overall success or failure of a method. Error records provide detailed information about the error, such as a text description of the error, the globally unique identifier (GUID) of the interface that defined the error, and provider-specific error information. Error objects in OLE DB are an extension of the error objects in Automation. They use many of the same mechanisms, and can be used as Automation error objects.

OLE DB error return codes are of type HRESULT. There are two general classes of return codes: success and warning codes, and error codes.

Success and warning codes begin with S_ or DB_S_ and indicate that the method successfully completed. The standard OLE DB error codes are defined in the OLEDBERR.H include file.

If the return code is other than S_OK or S_FALSE, it is likely that an error occurred from which the method was able to recover. For example, **IRowset::GetNextRows** returns DB_S_ENDOFROWSET when it is unable to return the requested number of rows due to reaching the end of the rowset. If a single warning condition occurs, the method returns the code for that condition. If multiple warning conditions occur, the method describes the hierarchy of warning return codes indicating which warning code should be returned when given a choice between multiple warning return codes.

Error codes begin with E_ or DB_E_ and indicate that the method failed completely and was unable to do any useful work. For example, **GetNextRows** returns E_INVALIDARG when the pointer in which it is to return a pointer to an array of row handles (*prghRows*) is null. An exception to this is that some of the methods that return DB_E_ERRORSOCCURRED allocate memory in which to return additional information about these errors. Consumers must free this memory. For information about which methods allocate memory in this case, see the methods that return DB_E_ERRORSOCCURRED. Although error codes can indicate run-time errors, such as running out of memory, they generally indicate programming errors. If multiple errors occur, the code that is returned is provider-specific. If both errors and warnings occur, the method fails and returns an error code.

All methods can return S_OK, E_FAIL, and E_OUTOFMEMORY. The E_OUTOFMEMORY code applies only to those methods which allocate memory that is returned to the consumer. In some cases, the E_OUTOFMEMORY code might be eliminated by calling the method requesting fewer returned values, such as fewer rows from **GetNextRows**.

Index Object (OLE DB Provider for AS/400 and VSAM)

An OLE DB index, also known as an **index rowset**, is a rowset built over an index in a data source. It is generally used in conjunction with a rowset built over a base table in the same data source. Each row of the index rowset contains a bookmark that points to a row in the base-table rowset. Thus, an OLE DB consumer can traverse the index rowset and use it to access rows in the base-table rowset.

Indexes are created using the index interfaces of the **Rowset** object. Index rowsets allow an application to read records efficiently by means of a key.

The following index interfaces of the **Rowset** object are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM when applied to AS/400 keyed physical files, AS/400 logical files, VSAM KSDS files with unique keys, and VSAM RRDS files with unique keys:

- **IAccessor**
- **IColumnsInfo**
- **IConvertType**
- **IRowset**
- **IRowsetChange**
- **IRowsetFind**
- **IRowsetIdentity**
- **IRowsetIndex**
- **IRowsetInfo**
- **IRowsetLocate**
- **IRowsetRefresh**
- **IRowsetScroll**
- **IRowsetUpdate**
- **IRowsetView**
- **ISupportErrorInfo**

OLE DB Provider for AS/400 and VSAM supports integrated indexes using the **IRowsetIndex** interface based on the underlying rowset.

Rowset Object (OLE DB Provider for AS/400 and VSAM)

Rowset objects are created by **Session** objects. The **Rowset** object exposes data in tabular format.

The following interfaces of the **Rowset** object are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM:

- **IAccessor**
- **IChapteredRowset**
- **IColumnsInfo**
- **IConvertType**
- **IRowset**
- **IRowsetChange**
- **IRowsetFind**
- **IRowsetIdentity**
- **IRowsetIndex**
- **IRowsetInfo**
- **IRowsetLocate**
- **IRowsetRefresh**
- **IRowsetUpdate**
- **IRowsetView**
- **ISupportErrorInfo**

Note
The **IRowsetFind** interface is only supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM when applied to AS/400 keyed physical files, AS/400 logical files, VSAM KSDS files with unique keys, and VSAM RRDS files with unique keys.

Session Object (OLE DB Provider for AS/400 and VSAM)

The **Session** object is created by a **DataSource** object. The **Session** object is used to create one or more **Rowset** objects.

The following interfaces of the **Session** object are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM:

- **IDBCreateCommand**
- **IDBSchemaRowset**
- **IGetDataSource**
- **IOpenRowset**
- **ISessionProperties**
- **ISupportErrorInfo**

Consumers can get information about a data store without knowing its structure by using the **IDBSchemaRowset** methods. The methods on this interface can be used to retrieve advanced schema information. OLE DB Provider for AS/400 and VSAM organizes this information into a set of schemas that contain tables for each schema. These schema rowsets are identified by globally unique identifiers (GUIDs).

The following schema rowset GUIDs are supported by OLE DB Provider for AS/400 and VSAM:

- DBSCHEMA_COLUMNS
- DBSCHEMA_INDEXES
- DBSCHEMA_PROVIDER_TYPES
- DBSCHEMA_TABLES

The following table lists these GUIDs and the columns for which restrictions can be specified on the schema rowset when using OLE DB Provider for AS/400 and VSAM. The number of restriction columns for each schema rowset are defined as constants prefixed with CRESTRICTIONS_ in the OLE DB header files. Restriction values are treated as literals rather than as search patterns. For example, the restriction value "A_C" matches "A_C" but not "ABC".

GUID	Number of restrictions	Restriction columns
DBSCHEMA_COLUMNS	4	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
DBSCHEMA_INDEXES	5	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME
DBSCHEMA_PROVIDER_TYPES	2	DATA_TYPE BEST_MATCH
DBSCHEMA_TABLES	4	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE

The **IDBSchemaRowset** interface allows an application to pass at run time the target library of a partitioned data set (PDS), a dataset, or a member name when using the **IDBSchemaRowset:GetSchemas** function to retrieve the schema.

This following example illustrates using a target library to retrieve a table schema:

```
hr = pIDBSchemaRowset->GetRowset(
        NULL,                          // punkOuter
        DBSCHEMA_TABLES,               // schema IID
         2L,                           // # of restrictions
```

```
        rgRestrictions,             // array of restrictions
        IID_IRowset,                // rowset interface
        0L,                         // # of properties
        NULL,                       // properties
        (IUnknown**)&pIRowset);     // rowset pointer
```

The variable *rgRestrictions* is an array containing two restriction values. The first array entry is VT_EMPTY and the second array entry is the target library name.

View Object (OLE DB Provider for AS/400 and VSAM)

The **View** object is created on a **Rowset** object. The **View** object is used to expose simple operations, such as sorting and filtering a rowset by applying a view. Views can be applied when opening a **Rowset** object or applied to an existing **Rowset** object.

The following interfaces of the **View** object are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM when applied to AS/400 keyed physical files, AS/400 logical files, VSAM KSDS files with unique keys, and VSAM RRDS files with unique keys:

- **IAccessor**
- **IColumnsInfo**
- **ISupportErrorInfo**
- **IViewChapter**
- **IViewFilter**
- **IViewRowset**
- **IViewSort**

OLE DB Property Support in the OLE DB Provider for AS/400 and VSAM

The following table summarizes the provider-specific OLE DB version 2.0 properties in the SNAOLEDB_DBPROPSET_DBINIT property set that are supported by the current version of Microsoft OLE DB Provider for AS/400 and VSAM.

OLE DB Property ID	Description
DBPROP_SNAOLEDB_APPCMODE	When logical unit (LU) 6.2 (SNA) is selected for the Network Transport Library (DBPROP_SNAOLEDB_NETTYPE), this property is the Advanced Program-to-Program Communications (APPC) mode and must be set to a value that matches the host configuration and SNA server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This VT_BSTR type property normally defaults to QPCSUPP.
DBPROP_SNAOLEDB_BINASCHAR	This property indicates whether to process binary fields (Character Code Set Identifier or CCSID of 65535) as character data type fields on a per data source basis. The host CCSID and PC Code Page values are required input parameters when this parameter is true. This VT_BOOL type property defaults to VARIANT_FALSE, do not process binary fields as character fields.
DBPROP_SNAOLEDB_HCDPATH	The fully qualified file name of the distributed data management (DDM) host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the SNA system directory. This VT_BSTR type property is required when connecting to mainframe systems and is optional when connecting to OS/400.
DBPROP_SNAOLEDB_HOSTCCSID	The CCSID matching the data as represented on the host. This property is required when processing binary data as character data. Unless the DBPROP_SNAOLEDB_BINASCHAR property is set to **true**, character data is converted based on the host column CCSID and default ANSI code page. This VT_I4 property defaults to U.S./Canada (37).
DBPROP_SNAOLEDB_LIBRARY	The default AS/400 library to be accessed. This VT_BSTR property is not required for mainframe access and is optional when connecting to AS/400 files.
DBPROP_SNAOLEDB_LOCALLU	When LU 6.2 (SNA) is selected for the Network Transport Library, this property is the name of the local LU alias configured in the SNA server.

DBPROP_SNAOLEDB_NETADDRESS	When TCP/IP has been selected for the Network Transport Library, this property is used to locate the target host computer. This parameter indicates the IP address or TCP/IP host name alias associated with the DDM server on the host. The network address is required when connecting through TCP/IP.
DBPROP_SNAOLEDB_NETPORT	When TCP/IP has been selected for the Network Transport Library, this property is used to locate the target DDM service access port when connecting by using TCP/IP. This parameter represents the TCP/IP port used for communication with the DDM service on the host. The default value for the VT_BSTR type property is 446.
DBPROP_SNAOLEDB_NETTYPE	This property which represents the dynamic link library used for transport designates whether the provider connects by using SNA LU 6.2 or TCP/IP for network communication. The possible values for this parameter are TCP/IP or SNA. If TCP/IP is selected, values for Network Address (DBPROP_SNAOLEDB_NETADDRESS) and Network Port (DBPROP_SNAOLEDB_NETPORT) are required. TCP/IP connectivity to the mainframe is not supported by OLE DB Provider for AS/400 and VSAM. If SNA is selected, values for APPC Local LU Alias (DBPROP_SNAOLEDB_LOCALLU), APPC Mode Name (DBPROP_SNAOLEDB_APPCMODE), and APPC Remote LU Alias (DBPROP_SNAOLEDB_REMOTELU) are required. This value for this VT_BSTR property defaults to SNA.
DBPROP_SNAOLEDB_PCCODEPAGE	The PC Code Page property ID indicates the code page to be used on the computer for character code conversion. This property is required when processing binary data as character data. Unless DBPROP_SNAOLEDB_BINASCHAR is set to true, character data is converted based on the default ANSI code page configured in Windows. If this parameter is set to Binary or 65535, no character code conversions will take place. The default value for this VT_I4 type property is 1252 (Latin-1).
DBPROP_SNAOLEDB_REMOTELU	When LU 6.2 (SNA) is selected for the Network Transport Library (DBPROP_SNAOLEDB_NETTYPE), this property ID is the name of the remote LU alias configured in the SNA server.
DBPROP_SNAOLEDB_REPAIRKEY	This property ID provides for repair of invalid key offsets received from OS/400 when keys have been defined using the Digital Data System (DDS) "RENAME" clause. This parameter indicates whether the OLE DB provider should repair any host key values set in the registry. This VT_BOOL type property defaults to VARIANT_FALSE.
DBPROP_SNAOLEDB_STRICTVAL	This property indicates whether strict validation should be used. This VT_BOOL type property defaults to VARIANT_FALSE.

OLE DB Object Support in the OLE DB Provider for DB2

The following table summarizes the OLE DB version 2.0 objects that are supported by the current version of Microsoft® OLE DB Provider for DB2.

OLE DB object	Support
Command Object	Yes, most interfaces
CustomErrorObject	Yes, all interfaces
DataSource Object	Yes, some interfaces
Enumerator	No
ErrorObject Object	Yes, all interfaces
ErrorRecord Object	Yes, all interfaces
Index	No
MultipleResults	No
Rowset Object	Yes, some interfaces
Session Object	Yes, some interfaces
Transaction Object	Yes, some interfaces
TransactionOptions	Yes, all interfaces
View	No

In This Section

- OLE DB Interface Support in the OLE DB Provider for DB2
- Command Object (OLE DB Provider for DB2)
- CustomErrorObject Object (OLE DB Provider for DB2)
- DataSource Object (OLE DB Provider for DB2)
- ErrorObject Object (OLE DB Provider for DB2)
- ErrorRecord Object (OLE DB Provider for DB2)
- Rowset Object (OLE DB Provider for DB2)
- Session Object (OLE DB Provider for DB2)
- Transaction Object (OLE DB Provider for DB2)
- OLE DB Property Support in the OLE DB Provider for DB2

OLE DB Interface Support in the OLE DB Provider for DB2

The following table summarizes the OLE DB version 2.0 interfaces that are supported by the current version of Microsoft OLE DB Provider for DB2.

Object	Interface	Support
Command Object	**IAccessor**	Yes
	IColumnsInfo	Yes
	IColumnsRowset	No
	ICommand	Yes
	ICommandPersist	No
	ICommandPrepare	Yes
	ICommandProperties	Yes
	ICommandText	Yes
	ICommandWithParameters	Yes
	IConvertType	Yes
	ISupportErrorInfo	Yes
CustomErrorObject Object	**IErrorLookup**	Yes
	ISQLErrorInfo	Yes
DataSource Object	**IDBAsynchStatus**	No
	IConnectionPointContainer	No
	IDBCreateSession	Yes
	IDBDataSourceAdmin	No
	IDBInfo	Yes
	IDBInitialize	Yes
	IDBProperties	Yes
	IPersist	Yes
	IPersistFile	Yes
	ISupportErrorInfo	Yes
Enumerator	**IDBInitialize**	No

	IDBProperties	No
	IParseDisplayName	No
	ISourcesRowset	No
	ISupportErrorInfo	No
ErrorObject Object	**IErrorRecords**	Yes
ErrorRecord Object	**IErrorInfo**	Yes
Index	**IAccessor**	No
	IColumnsInfo	No
	IConvertType	No
	IRowset	No
	IRowsetChange	No
	IRowsetFind	No
	IRowsetIdentity	No
	IRowsetIndex	No
	IRowsetInfo	No
	IRowsetLocate	No
	IRowsetRefresh	No
	IRowsetScroll	No
	IRowsetUpdate	No
	IRowsetView	No
	ISupportErrorInfo	No
MultipleResults	**IMultipleResults**	No
	ISupportErrorInfo	No
Rowset Object	**IAccessor**	Yes
	IChapteredRowset	No
	IColumnsInfo	Yes
	IColumnsRowset	No

	IConnectionPointContainer	No
	IConvertType	Yes
	IDBAsynchStatus	No
	IRowset	Yes
	IRowsetChange	Yes
	IRowsetChapterMember	No
	IRowsetFind	No
	IRowsetIdentity	No
	IRowsetIndex	No
	IRowsetInfo	Yes
	IRowsetLocate	No
	IRowsetRefresh	No
	IRowsetScroll	No
	IRowsetUpdate	Yes
	IRowsetView	No
	ISupportErrorInfo	Yes
Session Object	**IAlterIndex**	No
	IAlterTable	No
	IDBCreateCommand	Yes
	IDBSchemaRowset	Yes
	IGetDataSource	Yes
	IIndexDefinition	No
	IOpenRowset	Yes
	ISessionProperties	Yes
	ISupportErrorInfo	Yes
	ITableDefinition	No

	ITransaction	Yes
	ITransactionJoin	No
	ITransactionLocal	Yes
	ITransactionObject	Yes
Transaction Object	**IConnectionPointContainer**	No
	ISupportErrorInfo	Yes
	ITransaction	Yes
TransactionOptions	**ISupportErrorInfo**	Yes
	ITransactionOptions	Yes
View	**IAccessor**	No
	IColumnsInfo	No
	ISupportErrorInfo	No
	IViewChapter	No
	IViewFilter	No
	IViewRowset	No
	IViewSort	No

Command Object (OLE DB Provider for DB2)

The **Command** object is created by an OLE DB consumer, or by a service provider on behalf of a consumer. A **Command** object is used to execute a distributed data management (DDM)-specific command on a remote DDM server. The **Command** object currently supports executing Command Language commands on AS/400 DDM servers.

It is important not to confuse a command, which is an OLE COM object, and its command text, which is a string. Commands are generally used for data definition, such as creating a table or granting privileges, and data manipulation, such as updating or deleting rows. A special case of data manipulation using the **Command** object is the creation of rowsets based on DB2 tables. When using the command text with DB2/400 on the AS/400 computer, table names specified in a command are by default passed as uppercase. If a table name uses mixed case, the table name must be passed in a quoted string.

Before a consumer can use a command, it must determine if commands are supported. To do this, the consumer calls **QueryInterface** for **IDBCreateCommand** on a session. If this interface is exposed, the provider supports commands. To create a command, the consumer then calls **IDBCreateCommand::CreateCommand** on the session. A single session can be used to create multiple commands.

When the command is first created, it does not contain a command text. The consumer sets the command text with **ICommandText::SetCommandText**. Because the text command syntax is provider-specific, the consumer passes the globally unique identifier (GUID) of the syntax to use. For use with Microsoft OLE DB Provider for DB2, the GUID is DBGUID_DBSQL. Note that under OLE DB Provider for DB2, this GUID signifies that the text command is a superset of ANSI SQL. The level at which the provider supports ANSI SQL is specified by the DBPROP_SQLSUPPORT property. This property is a bitmask specifying the level of support for SQL.

The syntax supported by OLE DB Provider for DB2 for command text is as Entry-Level ANSI SQL 92 (with some exceptions based on the DB2 server host platform).

Legal SQL commands are documented in the following publications published by IBM:

- *AS/400 Advanced Series: DB2 for AS/400 SQL Reference Version 4* (Document Number SC41-5612-00)
- *DB2 for OS/390 Version 5: SQL Reference* (Document Number SC26-8966)

To execute the command, the consumer calls **ICommand::Execute**. If the command text specifies the command to open a rowset, **Execute** instantiates the rowset and returns an interface pointer to it.

The following interfaces of the **Command** object are supported by the current version of OLE DB Provider for DB2:

- **IAccessor**
- **IColumnsInfo**
- **ICommand**
- **ICommandPrepare**
- **ICommandProperties**
- **ICommandText**
- **ICommandWithParameters**
- **IConvertType**
- **ISupportErrorInfo**

When using the **ICommand** object, Microsoft OLE DB Provider for DB2 cannot derive parameter type information from the data store. The OLE DB client application must supply the native parameter type information through **ICommandWithParameters::SetParameterInfo** function. The OLE DB provider uses the type information specified by **SetParameterInfo** to determine how to convert parameter data from the type supplied by the consumer (as indicated by the

wType value in the binding structure) to the native type used by the data store. When the consumer specifies a data type with known precision, scale, and size values, any information supplied by the consumer for precision, scale, or size is ignored by OLE DB Provider for DB2.

The information that the consumer supplies must be correct and must be supplied for all parameters. OLE DB Provider for DB2 cannot verify the supplied information against the parameter metadata, although the OLE DB provider can determine that the specified values are legal values for the provider. The result of executing a command using incorrect parameter information or passing parameter information for the wrong number of parameters is undefined. For example, if the parameter type is **LONG** and the consumer specifies a type indicator of DBTYPE_STR in **ICommandWithParameters::SetParameterInfo**, OLE DB Provider for DB2 converts the data to a string before sending it to the data store. Because the data store is not expecting a **LONG**, this will likely result in an error.

CustomErrorObject Object (OLE DB Provider for DB2)

The **CustomErrorObject** object is created by a **Command** object when a command error occurs. The **CustomErrorObject** object is used to retrieve additional information when an error occurs.

The following interfaces of the **CustomErrorObject** object are supported by the current version of Microsoft OLE DB Provider for DB2:

- **IErrorLookup**
- **ISQLErrorInfo**

DataSource Object (OLE DB Provider for DB2)

The **DataSource** object is created by an OLE DB consumer. The **DataSource** object contains the knowledge and ability to connect to DB2 over Advanced Program-to-Program Communications (APPC) and LU 6.2 (through Host Integration Server 2009) or over TCP/IP. The **DataSource** object is used to create one or more **Session** objects.

The following interfaces of the **DataSource** object are supported by the current version of Microsoft OLE DB Provider for DB2:

- **IDBCreateSession**
- **IDBInfo**
- **IDBInitialize**
- **IDBProperties**
- **IPersist**
- **IPersistFile**
- **ISupportErrorInfo**

ErrorObject Object (OLE DB Provider for DB2)

The **ErrorObject** object is created by any interface on any DB2 OLE DB object. The **ErrorObject** object is used to retrieve additional information when an error occurs.

The following interface of the **ErrorObject** object is supported by the current version of Microsoft OLE DB Provider for DB2:

- **IErrorRecords**

The **IErrorRecords** interface returns **ErrorRecord** objects with detailed information on the error that occurred.

ErrorRecord Object (OLE DB Provider for DB2)

The **ErrorRecord** object is created by calling the **IErrorRecord** interface on the **ErrorObject** object. An **ErrorObject** is created on any interface on any SNA OLE DB object when an error occurs. The **ErrorRecord** object is used to retrieve additional information when an error occurs.

The following interface of the **ErrorRecord** object is supported by the current version of Microsoft OLE DB Provider for DB2:

- **IErrorInfo**

OLE DB interface methods return error information in two ways. The error code returned by an interface method, known as the **return code**, indicates the overall success or failure of a method. Error records provide detailed information about the error, such as a text description of the error, the globally unique identifier (GUID) of the interface that defined the error, and provider-specific error information. Error objects in OLE DB are an extension of the error objects in Automation. They use many of the same mechanisms, and can be used as Automation error objects.

OLE DB error return codes are of type HRESULT. There are two general classes of return codes: success and warning codes, and error codes.

Success and warning codes begin with S_ or DB_S_ and indicate that the method successfully completed. The standard OLE DB error codes are defined in the OLEDBERR.H include file.

If the return code is other than S_OK or S_FALSE, it is likely that an error occurred from which the method was able to recover. For example, **IRowset::GetNextRows** returns DB_S_ENDOFROWSET when it is unable to return the requested number of rows due to reaching the end of the rowset. If a single warning condition occurs, the method returns the code for that condition. If multiple warning conditions occur, the method describes the hierarchy of warning return codes, indicating which warning code should be returned when given a choice between multiple warning return codes.

Error codes begin with E_ or DB_E_ and indicate that the method failed completely and was unable to do any useful work. For example, **GetNextRows** returns E_INVALIDARG when a null pointer in which the OLE DB provider returns a pointer to an array of row handles (*prghRows*). An exception to this is that some of the methods that return DB_E_ERRORSOCCURRED allocate memory to return additional information about these errors. Consumers must free this memory. For information about which methods allocate memory in this case, see the methods that return DB_E_ERRORSOCCURRED.

Although error codes can indicate run-time errors, such as running out of memory, they generally indicate programming errors. If multiple errors occur, the code that is returned is provider-specific. If both errors and warnings occur, the method fails and returns an error code.

All methods can return S_OK, E_FAIL, and E_OUTOFMEMORY. The E_OUTOFMEMORY code applies only to those methods which allocate memory that is returned to the consumer. In some cases, the E_OUTOFMEMORY code might be eliminated by calling the method requesting fewer returned values, such as fewer rows from **GetNextRows**.

Rowset Object (OLE DB Provider for DB2)

Rowset objects are created by **Session** objects. The **Rowset** object exposes data in tabular format.

The following interfaces of the **Rowset** object are supported by the current version of Microsoft OLE DB Provider for DB2.

- **IAccessor**
- **IColumnsInfo**
- **IConvertType**
- **IRowset**
- **IRowsetChange**
- **IRowsetInfo**
- **IRowsetUpdate**
- **ISupportErrorInfo**

Session Object (OLE DB Provider for DB2)

The **Session** object is created by a **DataSource** object. The **Session** object is used to create one or more **Rowset** objects.

The following interfaces of the **Session** object are supported by the current version of Microsoft OLE DB Provider for DB2:

- **IDBCreateCommand**
- **IDBSchemaRowset**
- **IGetDataSource**
- **IOpenRowset**
- **ISessionProperties**
- **ISupportErrorInfo**
- **ITransaction**
- **ITransactionLocal**
- **ITransactionObject**

Consumers can get information about a data store without knowing its structure by using the **IDBSchemaRowset** methods. The methods on this interface can be used to retrieve advanced schema information. OLE DB Provider for DB2 organizes each DB2 database server in a set of schemas that contain tables for each schema. These schema rowsets are identified by globally unique identifiers (GUIDs).

The following schema rowset GUIDs are supported by OLE DB Provider for DB2:

- DBSCHEMA_COLUMNS
- DBSCHEMA_INDEXES
- DBSCHEMA_PRIMARY_KEYS
- DBSCHEMA_PROCEDURES
- DBSCHEMA_PROCEDURE_PARAMETERS
- DBSCHEMA_PROVIDER_TYPES
- DBSCHEMA_TABLES

The following table lists these GUIDs and the columns for which restrictions can be specified on the schema rowset when using OLE DB Provider for DB2. The number of restriction columns for each schema rowset are defined as constants prefixed with CRESTRICTIONS_ in the OLE DB header files. Restriction values are treated as literals rather than as search patterns. For example, the restriction value "A_C" matches "A_C" but not "ABC".

GUID	Number of restrictions	Restriction columns
DBSCHEMA_COLUMNS	4	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
DBSCHEMA_INDEXES	4	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TABLE_NAME

DBSCHEMA_PRIMARY_KEYS	3	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
DBSCHEMA_PROCEDURES	4	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PROCEDURE_TYPE
DBSCHEMA_PROCEDURE_PARAMETERS	4	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMETER_NAME
DBSCHEMA_PROVIDER_TYPES	2	DATA_TYPE BEST_MATCH
DBSCHEMA_TABLES	4	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE

Note that the TYPE restriction on the DBSCHEMA_INDEXES GUID is not supported by OLE DB Provider for DB2.

The PROCEDURE_SCHEMA restriction on the DBSCHEMA_PROCEDURE GUID and the DBSCHEMA_PROCEDURE_PARAMETERS GUID is not supported when connecting to DB/2 on OS/390 platforms.

Transaction Object (OLE DB Provider for DB2)

The **Transaction** object is created by a **Session** object. The **Transaction** object is used to manage transactions on one or more **Rowset** objects.

The following interfaces of the **Transaction** object are supported by the current version of Microsoft OLE DB Provider for DB2:

- **ISupportErrorInfo**
- **ITransaction**

The current implementation of OLE DB Provider for DB2 services all OLE DB **Session**, **Command**, and **Rowset** objects present in a given instance of the **DataSource** object through a single Advanced Program-to-Program Communications (APPC) conversation or TCP/IP connection. One implication of this design is that if two **Rowset** objects, each created from a different OLE DB **Session** object, use explicit commitment control through the **ITransaction** interface, they will interfere with each other. When a **Commit** or **Abort** for one instance is invoked, all work for the **DataSource** object will be either committed or aborted. This may yield undesirable results. The work around to this problem is to instantiate two instances of the **DataSource** object.

OLE DB Property Support in the OLE DB Provider for DB2

OLE DB Provider for DB2 included with Host Integration Server 2009 supports a different set of provider-specific properties than the earlier OLE DB Provider for DB2 supplied with SNA Server 4.0. The sections below provide information about provider-specific and standard OLE DB properties supported by the current and the previous version of OLE DB provider.

In This Section

OLE DB Provider-Specific Property Support in the OLE DB Provider for DB2

The following table summarizes the provider-specific OLE DB version 2.0 properties in the DB2OLEDB_DBPROPSET_DBINIT property set that are supported by the version of Microsoft OLE DB Provider for DB2 included with Host Integration Server 2009.

OLE DB Property ID	Description
DBPROP_DB2OLEDB_APPCMODE	When logical unit (LU) 6.2 (SNA) is selected for the Network Transport Library (DBPROP_DB2OLEDB_NETTYPE), this property is the Advanced Program-to-Program Communications (APPC) mode and must be set to a value that matches the host configuration and SNA server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This VT_BSTR type property normally defaults to QPCSUPP.
DBPROP_DB2OLEDB_BINASCHAR	This property indicates whether to process binary fields (Character Code Set Identifier or CCSID of 65535) as character data type fields on a per data source basis. The host CCSID and PC Code Page values are required input parameters when this parameter is true. This VT_BOOL type property defaults to VARIANT_FALSE, do not process binary fields as character fields.
DBPROP_DB2OLEDB_CATALOGCOL	The name of the collection where OLE DB Provider for DB2 looks for catalog information. This is the default schema, the "SCHEMA" name for the target collection of tables and views. This property is the Data Schema value when configuring data sources. OLE DB Provider for DB2 uses this default schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection. For DB2, the default schema is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the default schema is the target COLLECTION name. For DB2 Universal Database (UDB), the default schema is the SCHEMA name. If the user does not provide a VT_BSTR value for DBPROP_DB2OLEDB_CATALOGCOL, the OLE DB provider uses the USER_ID provided at logon. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Because these values for the default schema are inappropriate in many cases, it is essential that the Default Schema value in the data source be defined.
DBPROP_DB2OLEDB_HOSTCCSID	The CCSID matching the data as represented on the host. This property is required when processing binary data as character data. Unless the DBPROP_DB2OLEDB_BINASCHAR property ID is set to **true**, character data is converted based on the host column CCSID and default ANSI code page. This VT_I4 type property defaults to 37 (U.S./Canada).
DBPROP_DB2OLEDB_LOCALLU	When LU 6.2 (SNA) is selected for the Network Transport Library, this property is the name of the local LU alias configured in the SNA server. This VT_BSTR type property has no default value.

DBPROP_DB2OLEDB_NETADDRESS	When TCP/IP has been selected for the Network Transport Library, this property is used to locate the target host computer. This parameter indicates the IP address or TCP/IP host name alias associated with the DDM server on the host. The network address is required when connecting by using TCP/IP. This VT_BSTR type property defaults to SNA.
DBPROP_DB2OLEDB_NETPORT	When TCP/IP has been selected for the Network Transport Library, this property is used to locate the target DDM service access port when connecting by using TCP/IP. This parameter represents the TCP/IP port used for communication with the DDM service on the host. This VT_BSTR type property defaults to 446.
DBPROP_DB2OLEDB_NETTYPE	This property, which represents the dynamic-link library used for transport, designates whether the provider connects through SNA LU 6.2 or TCP/IP for network communication. The possible values for this parameter are TCPIP or SNA. If TCPIP is selected, values for Network Address (DBPROP_DB2OLEDB_NETADDRESS) and Network Port (DBPROP_DB2OLEDB_NETPORT) are required. If SNA is selected, values for APPC Local LU Alias (DBPROP_DB2OLEDB_LOCALLU, APPC), Mode Name (DBPROP_DB2OLEDB_APPCMODE), and APPC Remote LU Alias (DBPROP_DB2OLEDB_REMOTELU) are required. This VT_BSTR type property defaults to SNA.
DBPROP_DB2OLEDB_PACKAGECOL	The name of the Distributed Relational Database Architecture (DRDA) target collection (AS/400 library) where Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be the same as the Default Schema (DBPROP_DB2OLEDB_DEFAULTSCH). Microsoft OLE DB Provider for DB2, which is implemented as an IBM DRDA Application Requester, uses packages to issue dynamic and static SQL statements. OLE DB Provider for DB2 will create packages dynamically in the location to which the user points using this property ID. This VT_BSTR type property has no default value.
DBPROP_DB2OLEDB_PCCODEPAGE	The PC Code Page property ID indicates the code page to be used on the computer for character code conversion. This property is required when processing binary data as character data. Unless DBPROP_DB2OLEDB_BINASCHAR is set to true, character data is converted based on the default ANSI code page configured in Windows. If this parameter is set to Binary or 65535, no character code conversions will take place. This VT_I4 type property defaults to 1252 (Latin 1).
DBPROP_DB2OLEDB_PLATFORM	The target DB2 platform property value is used to optimize performance of the OLE DB provider when executing operations such as data conversion. The following values for this property are supported by OLE DB Provider for DB2: ● DB2/MVS ● DB2/NT ● DB2/6000 ● DB2/400 This VT_BSTR property has a default value of DB2/MVS.

DBPROP_DB2OLEDB_QUALIFIERCOL	The name of the schema (collection/owner) used to fully qualify unqualified object names. Note that this attribute allows the user to access database objects without fully qualifying the object using a collection (schema) qualifier. The OLE DB provider sends this value to DB2 using a **SET CURRENT SQLID** statement, instructing th e DBMS to use this value when locating unqualified objects (for example, tables and views) referenced in SQL statemen ts. If you do not set a value for default qualifier, no **SET** statement is issued by the ODBC driver. This OLE DB property is only valid when connecting to DB2 for MVS (OS/390, z/OS). This VT_BSTR type property has a default value of .
DBPROP_DB2OLEDB_REMOTELU	When LU 6.2 (SNA) is selected for the Network Transport Library (DBPROP_DB2OLEDB_NETTYPE), this property is the name of the remote LU alias configured in the SNA server. This VT_BSTR type property has no default value.
DBPROP_DB2OLEDB_TPNAME	This property represents the default transaction program (TP) name for the DB2 DRDA application server (AS), which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, t his property is set to 0X07F9F9F9. This VT_BSTR type property has no default value.
DBPROP_DB2OLEDB_UNITSOFWORK	This property indicates whether two-phase commit (distributed unit of work) used for transactions is supported for thi s data source. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transacti on Coordinator, and SNA LU 6.2 Resync Service. The following values for this property are supported by OLE DB Provider for DB2: ● Remote unit of work (RUW) ● Distributed unit of work (DUW) This VT_BSTR type property has a default value of RUW. Distributed unit of work (two-phase commit) works only with DB2 for OS/390 v5R1 or later. This option also requires t hat the SNA LU 6.2 service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

OLE DB Data Source Property Support in the OLE DB Provider for DB2

The following table summarizes the standard OLE DB version 2.0 data source properties from the DBPROP_DATASOURCE property set that are supported by the version of Microsoft OLE DB Provider for DB2 included with Host Integration Server 2009.

OLE DB Property ID	Comments
DBPROP_CURRENTCATALOG	The name of the current catalog. This property is derived from the *Initial Catalog* parameter when configuring data sources. An application can use the CATALOGS schema rowset to enumerate catalogs. This VT_BSTR type property defaults to the value configured in the data source for *Initial Catalog*.

ODBC Driver for DB2 Programmer's Reference

The Microsoft ODBC Driver for DB2 enables users to access IBM DB2 from within an ODBC-aware application. ODBC defines a standard set of interfaces that provide access to disparate databases. This section provides reference material needed to program for the ODBC Driver for DB2.

For general information about programming for the ODBC Driver for DB2, see the ODBC Driver for DB2 Programmer's Guide section of the SDK.

In This Section

Programming Considerations When Using the ODBC Driver for DB2

ODBC Conformance

ADO Object, Method, Property, and Collection Support for AS/400, VSAM and DB2

Programming Considerations When Using the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2 provides passthrough support for SQL statements. No SQL parsing is provided. The user must know what SQL syntax is supported for the target DB2-implementation. For information on what SQL syntax is supported, see the specific DB2 SQL Reference and DB2 Application Programming and SQL Guide for the specific platform.

The ODBC Driver for DB2 does not parse the SQL statements to qualify table names. Consequently, users of the ODBC Driver for DB2 must use either two-part or three-part (fully qualified) object names when naming tables, views, and stored procedures in DB2. A two-part table name would consist of the user ID and table, <UserID>.<Table>. One-part names (just the table name) will not succeed unless the combination of the DB2 collection and schema name correspond directly to the ODBC User ID

The Microsoft ODBC Driver for DB2 does not insert the correct value for the fraction when using a parameterized insert with the TIMESTAMP data type.

Microsoft Data Access Components (MDAC) supports the option of using a client cursor engine. This service component is implemented as part of OLE DB, ADO, and Remote Data Services (RDS). When using ADO, a client cursor is enabled by setting the **CursorLocation** property on the recordset to **adUseClient**. When using the ADO Client Cursor Engine with DB2 for OS/390, the developer must configure the ODBC Driver for DB2 Auto Commit Mode property in the DSN or connection string to **FALSE**. This is not required when connecting to DB2 for OS/400.

The ODBC Driver for DB2 included with Host Integration Server 2009 supports updating capabilities when used with a client cursor engine and the following requirements are met:

- To support updates (UPDATE, INSERT, and DELETE), the values in at least one column in the target table must be unique.
- The Auto Commit parameter must be set to **FALSE** when configuring the data source or when this parameter is passed as part of a connection string.

Previous versions of the ODBC Driver for DB2 do not support any updating capabilities when used with a client cursor engine. In other words, if a client cursor engine is enabled using RDS or ADO, the ODBC Driver for DB2 cannot be used to update data on the host. The ADO recordset is treated as if it were read-only. When using the ADO Client Cursor Engine with DB2 for OS/390, the developer must configure the ODBC Driver for DB2 Auto Commit parameter in the data source or connection string to FALSE. This is not required when connecting to DB2 for OS/400.

When the intent is to update records with a server-side cursor, DB2 requires that the SQL **SELECT** statement also include the FOR UPDATE option. For example, to select all records from the AUTHORS table in the DB2 collection called PUBS with intent to update requires the following SQL syntax:

```
SELECT * FROM PUBS.AUTHORS FOR UPDATE
```

When using DB2 for MVS V4R1 and DB2 for OS/400 V3R2, there are further requirements to indicate the columns that you intend to update. For example, to update the AU_LNAME and AU_FNAME columns in the PUBS.AUTHORS table, the following SQL syntax must be used:

```
SELECT * FROM PUBS.AUTHORS FOR UPDATE OF AU_LNAME, AU_FNAME
```

The Microsoft ODBC Driver for DB2 provides support for distributed transactions and DRDA Distributed Unit of Work, and can participate in a distributed transaction coordinated by Microsoft Distributed Transaction Coordinator. This feature is only available when connecting to one of the following across an LU 6.2 network connection:

- DB2 for OS/390 V5R1 or later.
- DB2 for DB2/400 V4R3 or later.

This option also requires that the SNA LU 6.2 service is selected as the network transport and Microsoft Transaction Server (MTS) is installed. The Microsoft ODBC Driver for DB2 does not support automatic transaction enlistment under Microsoft Transaction Server.

Applications should not commit or roll back transactions by executing **COMMIT** or **ROLLBACK** statements using the **SQLExecute** or **SQLExecDirect** ODBC functions. The effects of doing this are undefined, and the ODBC Driver for DB2 no longer knows when a transaction is active. Applications should call the **SQLEndTran** ODBC function instead.

Microsoft Visual Studio offers a number of ADO data-bound controls, including a data grid and the ADO Data control. When using these ADO data controls, the developer must set the **CursorLocation** property on the recordset to **adUseClient**. Additionally, when using these ADO data controls with DB2 for OS/390, you must set the ODBC Driver for DB2 Auto Commit parameter in the data source or connection string to **FALSE**.

You can use Microsoft Query (MSQUERY) supplied with Microsoft Office Excel and Microsoft Office to access ODBC data sources using the ODBC Driver for DB2. When a data of a column defined with TIMESTAMP data type is updated using Microsoft Query, the microseconds portion of the TIMESTAMP is not updated properly. The result is that updating any TIMESTAMP value using Microsoft Query will cause the loss of fractional seconds.

The Microsoft ODBC Driver for DB2 is not supported for use with the Microsoft ODBC .NET Framework Data Provider. When accessing DB2 from ADO.NET, use the Microsoft OLE DB Provider for DB2 in conjunction with the Microsoft OLE DB .NET Framework Data Provider.

In This Section

Stored Procedure Support Using the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2 supports calling DB2 stored procedures, returning output parameters and single and multiple result sets.

Support for Isolation Levels Using the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2 provides flexibility in dealing with issues of isolation levels and transaction state. The ODBC **SQLSetConnectAttr** function is used to set the isolation level that is to be used for a connection. This function would be called with the attribute parameter set to SQL_ATTR_TXN_ISOLATION and the *ValuePtr* parameter pointing to an integer value indicating the isolation level requested. This integer value is a 32-bit bitmask that sets the transaction isolation level for the current connection.

The allowable values for isolation level (the *ValuePtr* parameter when calling **SQLSetConnectAttr**) can be determined by calling **SQLGetInfo** with InfoType equal to SQL_TXN_ISOLATION_OPTION. The following table list the allowable values for isolation level using the ODBC Driver for DB2 supplied with Host Integration Server.

ODBC Isolation Level Attribute	**Description**
SQL_TXN_READ_COMMITTED	When this attribute value is set, it isolates any data read from changes by others and changes made by others by others cannot be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read. This is the default value for isolation level This isolation level is also called Cursor Stability (CS) in IBM DB2 documentation.
SQL_TXN_READ_UNCOMMITTED	When this attribute value is set, it does not isolate data read from changes by others and changes made by others by others can be seen. The re-execution of the read statement is affected by others. This does not support a repeatable read. This isolation level is called Uncommitted Read (UR) in IBM DB2 documentation.
SQL_TXN_REPEATABLE_READ	When this attribute value is set, it isolates any data read from changes by others and changes made by others cannot be seen. The re-execution of the read statement is affected by others. This supports a repeatable read. This isolation level is called Read Stability (RS) in IBM DB2 documentation.
SQL_TXN_SERIALIZABLE	When this attribute value is set, it isolates any data read from changes by others and changes made by others by others cannot be seen. The re-execution of the read statement is not affected by others. This supports a repeatable read. This isolation level is called Repeatable Read (RR) in IBM DB2 documentation.

The SQL_ATTR_TXN_ISOLATION attribute can be set only if there are no open transactions on the connection. An application must call **SQLEndTran** to commit or roll back all open transactions on a connection, before calling **SQLSetConnectAttr** with this option.

Some connection attributes support substitution of a similar value if the data source does not support the value specified in *ValuePtr*. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). To determine the substituted value, an application calls **SQLGetConnectAttr**.

Code Page Support Using the ODBC Driver for DB2

When creating data sources or file DSNs for use with the ODBC Driver for DB2, the Host character code set identifier (CCSID) should be configured in the data source to match the DB2 data as represented on the remote host computer. The Host CCSID parameter defaults to EBCDIC U.S./Canada (37).

On Windows 2000, the appropriate ANSI NLS file for your locale is installed automatically when you install a localized version of Windows 2000.

The following sections discuss the character code set identifiers (CCSIDs) supported by ODBC Driver for DB2 in Host Integration Server 2009.

In This Section

ANSI Code Page Support Using the ODBC Driver for DB2

IBM DB2 Universal Database for Windows NT and IBM DB2 Universal Database for AIX are frequently configured to use ANSI code pages, for example ANSI 1253 (Greek). Host Integration Server 2009 includes support for some ANSI code pages for purposes of ANSI-to-UNICODE-to-ANSI conversions when using the OLE DB Provider for DB2 or the ODBC Driver for DB2. These ANSI code pages can be used when accessing IBM DB2 Universal Database on Windows NT and IBM DB2 ON AIX (not all of these ANSI code pages are supported on IBM DB2 Universal Database for AIX).

The following table shows the ANSI character code set identifiers (CCSIDs) supported by ODBC Driver for DB2 in Host Integration Server 2009.

Microsoft Display Name	Microsoft NLS Code Page	IBM CCSID	Comments
ANSI - Arabic	1256	1256	
ANSI - Baltic	1257	1257	
ANSI - Cyrillic	1251	1251	
ANSI - Central Europe	1250	1250	
ANSI - Greek	1253	1253	
ANSI - Hebrew	1255	1255	
ANSI - Latin I	1252	1252	Support for this codepage is normally installed as part of the operating system on Windows 2000.
ANSI - Turkish	1254	1254	
ANSI/OEM - Japanese Shift JIS	932	932	
ANSI/OEM - Korean	949	949	
ANSI/OEM - Simplified Chinese GBK	936	936	
ANSI/OEM - Thai	874	874	
ANSI/OEM - Traditional Chinese Big5	950	950	
ANSI/OEM - Viet Nam	1258	1258	

The Microsoft Display Name is the name found in the Windows 2000 definitions for these NLS files. The Microsoft NLS Code Page column represents the code page number that is registered and associated with an ANSI-to-UNICODE NLS resource file. The Microsoft NLS number should be set as the Host CCSID when configuring data sources when using the ODBC Driver for DB2. When setting the Host CCSID or PC Code Page parameter using a connection string, the Microsoft NLS number should be used for this parameter.

The IBM CCSID column represents the CCSID given to the ANSI code page in IBM publications, which for these supported ANSI CCSIDs are the same as the Microsoft CCSID values. IBM lists their ANSI support in publications by referencing the display name which for these ANSI code pages is the same as the Microsoft display name. The ODBC Driver for DB2 does not recognize or display the IBM CCSID values when configuring data sources. The ODBC Driver for DB2 maps the Microsoft NLS numbers to ANSI NLS files which correspond with the appropriate IBM CCSID numbers. The Microsoft ODBC Driver for DB2 passes the corresponding IBM CCSID to the DB2 system at run time even though you configure the driver to use the Microsoft

NLS number.

These are the only ANSI pages currently supported by the ODBC Driver for DB2 in Host Integration Server 2009 and in SNA Server 4.0 with Service Pack 3 or later. IBM supports additional ANSI pages, however, the ANSI code pages listed in the table above are the only cases where the Microsoft NLS pages and IBM ANSI code pages (CCSIDs) match.

EBCDIC Code Page Support Using the ODBC Driver for DB2

IBM DB2 for MVS, IBM DB2 for OS/390, and IBM DB2 for OS/400 are frequently configured to use EBCDIC code pages, for example EBCDIC 875 (Greek Modern). Host Integration Server 2009 includes support for most EBCDIC code pages for purposes of EBCDIC-to-UNICODE-to-ANSI, ANSI-to-UNICODE-to-EBCDIC, and EBCDIC-to-UNICODE-to-EBCDIC conversions when using the OLE DB Provider for DB2 or the ODBC Driver for DB2. These EBCDIC code pages can be used when accessing IBM DB2 on a variety of platforms (not all of these EBCDIC code pages are supported on all versions of IBM DB2).

The following table shows the EBCDIC character code set identifiers (CCSIDs) supported by ODBC Driver for DB2 in Host Integration Server 2009.

Microsoft Display Name	Microsoft NLS Code Page	IBM CCS ID	Comments
IBM EBCDIC - Arabic	20420	420	
IBM EBCDIC - Cyrillic (Russian)	20880	880	
IBM EBCDIC - Cyrillic (Serbian, Bulgarian)	21025	1025	
IBM EBCDIC - Denmark/Norway	20277	277	
IBM EBCDIC - Denmark/Norway (Euro)	1142	1142	
IBM EBCDIC - Finland/Sweden	20278	278	
IBM EBCDIC - Finland/Sweden (Euro)	1143	1143	
IBM EBCDIC - France	20297	297	
IBM EBCDIC - France (Euro)	1147	1147	
IBM EBCDIC - Germany	20273	273	
IBM EBCDIC - Germany (Euro)	1141	1141	
IBM EBCDIC - Greek	20423	423	
IBM EBCDIC - Greek (Modern)	875	875	
IBM EBCDIC - Hebrew	20424	424	
IBM EBCDIC - Icelandic	20871	871	
IBM EBCDIC - Icelandic (Euro)	1149	1149	
IBM EBCDIC - International	500	500	
IBM EBCDIC - International (Euro)	1148	1148	
IBM EBCDIC - Italy	20280	280	
IBM EBCDIC - Italy (Euro)	1144	1144	

IBM EBCDIC - Japan English/Kanji (Extended)	939	939	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Japan English/Kanji (Extended)	5035	5035	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Japan Katakana/Kanji (Extended)	930	930	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Japan Katakana/Kanji (Extended)	5026	5026	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Japanese	931	931	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Korea (Extended)	933	933	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Latin America/Spain	20284	284	
IBM EBCDIC - Latin America/Spain (Euro)	1145	1145	
IBM EBCDIC - Multilingual/ROECE (Latin-2)	870	870	
IBM EBCDIC - Simplified Chinese (Extended)	935	935	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Thai	20838	838	
IBM EBCDIC - Traditional Chinese (Extended)	937	937	Support for this double-byte character set is supplied using TRNSDT.
IBM EBCDIC - Turkish (Latin-3)	20905	905	
IBM EBCDIC - Turkish (Latin-5)	1026	1026	
IBM EBCDIC - U.S./Canada	037	37	
IBM EBCDIC - U.S./Canada (Euro)	1140	1140	
IBM EBCDIC - United Kingdom	20285	285	
IBM EBCDIC - United Kingdom (Euro)	1146	1146	

The Microsoft Display Name is the name found in the Windows 2000 definitions for these NLS files. The Microsoft NLS Code Page column represents the code page number that is registered and associated with an EBCDIC-to-UNICODE NLS resource file. The Microsoft NLS number should be set as the Host CCSID when configuring data sources when using the ODBC Driver for DB2. When setting the Host CCSID or PC Code Page parameter using a connection string, the Microsoft NLS number should be used for this parameter.

The IBM CCSID column represents the CCSID given to the EBCDIC code page in IBM publications. IBM lists their EBCDIC support in publications by referencing the display name which for these EBCDIC code pages is the same as the Microsoft display name. The ODBC Driver for DB2 does not recognize or display the IBM CCSID values when configuring data sources using data links. The ODBC Driver for DB2 maps the Microsoft NLS numbers to EBCDIC NLS files which correspond with the

appropriate IBM CCSID numbers. The Microsoft ODBC Driver for DB2 passes the corresponding IBM CCSID to the DB2 system at run time even though you configure the driver to use the Microsoft NLS number.

These are the only EBCDIC pages currently supported by the ODBC Driver for DB2 in Host Integration Server 2009 and in SNA Server 4.0 with Service Pack 3 or later. IBM supports additional EBCDIC pages; however, the EBCDIC code pages listed in the table above are the only cases where the Microsoft NLS pages and IBM EBCDIC code pages (CCSIDs) match.

ISO Code Page Support Using the ODBC Driver for DB2

IBM DB2 Universal Database for Windows NT and IBM DB2 Universal Database for AIX are frequently configured for an ISO code page, for example ISO 819 (Latin I). Host Integration Server 2009 includes support for some ISO code pages for purposes of ISO-to-UNICODE-to-ANSI, ANSI-to-UNICODE-to-ISO, and ISO-to-UNICODE-to-ISO conversions when using the OLE DB Provider for DB2 or the ODBC Driver for DB2. These ISO code pages can be used when accessing IBM DB2 Universal Database for Windows NT and IBM DB2 Universal Database for AIX.

The following table shows the ISO character code set identifiers (CCSIDs) supported by ODBC Driver for DB2 in Host Integration Server.

Microsoft Display Name	Microsoft NLS Code Page	IBM CCSID	Comments
ISO 8859-1 Latin 1	28591	819	
ISO 8859-2 Central Europe	28592	912	
ISO 8859-5 Cyrillic	28595	915	
ISO 8859-6 Arabic	28596	1089	
ISO 8859-7 Greek	28597	813	
ISO 8859-8 Hebrew	28598	916	
ISO 8859-9 Turkish	28599	920	
ISO 6937 Non-Spacing Accent	20269	819	Note that ISO 6937 (CCSID 20269) is not supported by the ODBC Driver for DB2, but is displayed in the list of configuration options when creating or modifying data sources.
ISO 8859-15 Latin 9 (Euro)	20865	923	NLS Code Page 819 with support for the euro.

The ///Microsoft Display Name is the name found in the Windows 2000 definitions for these NLS files.

The Microsoft NLS Code Page column represents the code page number that is registered and associated with an ISO-to-UNICODE NLS resource file. The Microsoft NLS number should be set as the Host CCSID when configuring data sources when using the ODBC Driver for DB2. When setting the Host CCSID or PC Code Page attribute/property using a connection string, the Microsoft NLS number should be used for this parameter.

The IBM CCSID column represents the CCSID given to the ISO code page in IBM publications. IBM lists their ISO support in publications by referencing the locale name (Bulgaria for ISO 8859-5 and 915, for example) rather than simply using ISO 8859-5 Cyrillic as used by Microsoft. The ODBC Driver for DB2 does not recognize or display the IBM CCSID values when configuring data sources. The ODBC Driver for DB2 maps the Microsoft NLS numbers to ISO NLS files which correspond with the appropriate IBM CCSID numbers. The Microsoft ODBC Driver for DB2 passes the corresponding IBM CCSID to the DB2 system at run time even though you configure the driver to use the Microsoft NLS number.

Note that IBM CCSID 819 is associated with both ISO 8859-1 Latin 1 and ISO 6937 Non-Spacing Accent. It is up to the user to choose the standard ISO 8859-1 Latin 1 code page by selecting NLS code page 28591 or the modified code page ISO 6937 Non-Spacing Accent by selecting NLS code page 20269. Note that ISO 6937 Non-Spacing Accent (CCSID 20269) is not currently supported by the ODBC Driver for DB2, but is displayed in the configuration options when creating or modifying data

sources.

IBM CCSID 916 (ISO 8859-8) supports Hebrew "visual sort order". IBM CCSID 920 (ISO 8859-8 derivation) supports Hebrew "logical sort order". Although Microsoft supports the logical sort order with NLS 38598, this NLS file is only distributed with Internet Explorer 5 or Windows 2000. The ODBC Driver for DB2 has not been tested using the ISO 8859-8 derivation matching IBM CCSID 920 and does not support this configuration.

These are the only ISO pages currently supported in Host Integration Server and in SNA Server 4.0 with Service Pack 3 or later. Microsoft supports a number of additional ISO pages. IBM also supports additional ISO pages. However, the code pages listed in the table above are the only cases where the Microsoft NLS pages and IBM CCSIDs match.

DBCS Code Page Support Using the ODBC Driver for DB2

The ODBC Provider for DB2 supports Double-Byte Character String (DBCS) data, including conversions between DBCS and ANSI code pages. Conversions between DBCS and ISO code pages are also supported, as are positioned updates against DBCS EBCDIC implementations of DB2.

The DB2 GRAPHIC data types (GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC) are now explicitly supported. These DB2 data types support DBCS (not mixed) data. Mixed data types are supported using CHAR FOR MIXED DATA, VARCHAR FOR MIXED DATA, and LONGVARCHAR FOR MIXED DATA.

Parameterized SQL statements and calling stored procedures when the parameter values contain Mixed or DBCS characters are supported.

When performing a SQL statement with inline data, character strings can be prefixed with a G or g to denote that the data is DBCS and is to be used with a graphic column. Only double-byte characters can be used with a graphic column. For example:

```
CREATE TABLE NAMES (COL1 GRAPHIC(64), COL2 CHAR(64))
INSERT INTO NAMES (COL1, COL2) VALUES (GDBCS data, MBCS or SBCS data)
```

The prefixes n (graphic) and gx (hexadecimal graphic) are also supported.

Data Conversion Using the ODBC Driver for DB2

The design of ODBC APIs is similar to other ISAM APIs. The APIs are handle-based. After opening a file, the application can determine the buffer size required to store a row, use the cursor APIs to move, and optionally retrieve one or more rows of data using the row-level binding.

Data is converted to default SQL data types, as defined in ODBC. The following table lists these conversions.

DB2 data type	Default SQL Data Type	Exposed as Native Type in SQLGetTypeInfo	Comments
BIGINT			An 8-byte integer. This data type is supported on DB2 UDB only.
BLOB			A Binary Large Object (BLOB) is a varying-length string that can be up to 2 gigabytes in length. A BLOB is primarily intended to hold binary data. This data type is not supported by the ODBC Driver for DB2.
CHAR (BIT)	SQL_BINARY	No	A fixed length (double-byte only) character string.
CHAR (SBCS)	SQL_CHAR	Yes	A fixed-length SBCS character string.
CHAR (MIXED)	SQL_CHAR	No	A fixed-length mixed (single and double-byte) character string.
CLOB			A Character Large Object(CLOB) is a varying-length string that can be up to 2 gigabytes in length. A CLOB is used to store large single-byte character set data. A CLOB is considered to be a character string. This data type is not supported by the ODBC Driver for DB2.
DATE	SQL_TYPE_DATE	Yes	A 10-byte date string. This data type is converted to an SQL_DATE for use by ODBC.
DBCLOB			A Double-Byte Character Large Object (DBCLOB) is a varying-length string of double-byte characters that can be up to 2 gigabytes in length (1,073,741,823 double-byte characters). A DBCLOB is used to store large double-byte character set data. A DBCLOB is considered to be a graphic string. It is not supported by the ODBC Driver for DB2.
DECIMAL	SQL_DECIMAL	Yes	A packed decimal number.
DOUBLE	SQL_DOUBLE	Yes	An 8-byte double-precision floating point number.
FLOAT	SQL_FLOAT	Yes	An 8-byte double-precision floating point number. This data type is the same as a DOUBLE.

GRAPHIC (DBCS)	SQL_CHAR	No	A fixed-length graphic string consisting of a sequence of double byte (DBCS only) character string data.
INTEGER	SQL_INTEGER	Yes	A 4-byte integer with a precision of 10 digits ranging in value from -2,147,463,648 to +2,147,483,647.
LONG VARCHAR (BIT)	SQL_BINARY	No	A varying-length (double-byte only) character string.
LONG VARCHAR (SBCS)	SQL_CHAR	No	A varying-length SBCS character string.
LONG VARCHAR (MIXED)	SQL_CHAR	No	A varying-length mixed-character (single and double-byte) string.
LONG VARGRAPHIC (DBCS)	SQL_LONGVARCHAR	No	A varying-length graphic string consisting of a sequence of double byte (DBCS only) character string data.
SMALLINT	SQL_SMALLINT	Yes	A SMALLINT (small integer) is a two-byte integer with a precision of 5 digits ranging from -32,768 to +32,767.
REAL	SQL_REAL	Yes	A 4-byte single-precision floating point number.
TIME	SQL_TYPE_TIME	Yes	An 8-byte time string. When using ActiveX Data Objects to return data from a DB2 TIME data type, ADO returns a DATETIME value.
TIMESTAMP	SQL_TYPE_TIMESTAMP	Yes	A 26-byte string representing the date, time, and microseconds.
VARCHAR (BIT)	SQL_BINARY	No	A varying-length (double-byte only) character string.
VARCHAR (SBCS)	SQL_CHAR	Yes	A varying-length character string.
VARCHAR (MIXED)	SQL_CHAR	No	A varying-length mixed (single and double-bye) character string.
VARGRAPHIC (DBCS)	SQL_VARCHAR	No	A varying-length graphic string consisting of a sequence of double byte (DBCS only) character string data.

Not all platforms and versions of DB2 support all of the above-referenced data types. Consult your IBM SQL Reference for the specific target and platform and version of DB2.

The ODBC Driver for DB2 exposes only selected DB2 data types as native types in the ODBC catalog function **GetTypeInfo**. For example, the driver does not expose LONG CHARACTER or VARYING LONG CHARACTER types. Rather these types are exposed as CHARACTER and VARYING CHARACTER respectively. Also, the driver exposes CHARACTER FOR SBCS DATA, CHARACTER FOR MIXED DATA, and CHARACTER FOR BIT DATA as CHARACTER. The driver exposes VARYING CHARACTER FOR SBCS DATA, VARYING CHARACTER FOR MIXED DATA, and VARYING CHARACTER FOR BIT DATA as VARYING CHARACTER. However, the ODBC Driver for DB2 returns these LONG and VARYING LONG data types if one reads a table with these data types. For example, when reading a table with a variable character string of length greater than 254 bytes, the ODBC Driver for DB2 returns a LONG VARCHAR.

The maximum length of the DB2 character and graphic string data types is dependent on the DB2 platform and version. For example, a CHAR type on DB2 for OS/390 V5R1 has a maximum length of 254 bytes, whereas a CHAR type on DB2/400 V4R4 has a maximum length of 32,766 bytes.

Data conversions from a large numeric type to a small numeric type are supported (from DOUBLE to SINGLE and from INT to SMALLINT, for example), however truncation and conversion errors can occur that will not be reported by the ODBC Driver for DB2.

For more information on support for the DB2 character data types of subtype MIXED using the ODBC Driver for DB2, see Code Page Support Using the ODBC Driver for DB2.

Using the ODBC Driver for DB2, certain conversions of strings from EBCDIC to ASCII and then back to EBCDIC are asymmetric, and can result in strings that are different from the original. The EBCDIC specification contains ordinals for which there is no defined character. The ODBC Driver for DB2 translates all such undefined characters to the question mark character ("?"). So when ASCII strings containing these characters are converted back to EBCDIC, these undefined characters are replaced with question marks. To protect EBCDIC strings containing undefined characters, these fields should be tagged as binary strings and mapped by the application.

The affected ANSI-to-EBCDIC character conversions include the following:

Character value (decimal)	Character value (hexadecimal)	ANSI code page 1252	EBCDIC character after conversion to CCSID 37
128	0x80	Not used	?
130	0x82	Single low quote	?
131	0x83	Latin F with hook	?
132	0x84	Double low quote	?
133	0x85	Ellipsis	?
134	0x86	Dagger	?
135	0x87	Double dagger	?
136	0x88	Per mile	?
137	0x89	S with caron	?
138	0x8A	Left angle	?
139	0x8B	Ligature OE	?
140	0x8C	Not used	?
142	0x8E	Not used	?

145-156	0x91-0x9C		?
158-159	0x9E-0x9F		?

Floating Point Considerations Using the ODBC Driver for DB2

When real or double (synonymous with float) data is inserted into a DB2 table as a floating point data type, it is stored in scientific notation. For example, FLOAT(1.1) would be stored as +1.10000E+000.

Care must be taken when executing SQL statements to make sure that the proper data type specified in the SQL statement matches the values stored in DB2. For example, the following select statement would match values in DB2 stored as decimal 1.1:

```
SELECT * FROM TEST WHERE C1 = 1.1
```

If the data in DB2 was stored as real numbers, there would not be a match since decimal 1.1 is stored as 1.1, not the representation of +1.10000E+000. When DB2 parses and executes the SQL **SELECT** statement, it interprets 1.1 as a decimal type. When doing the **SELECT** query, DB2 does not implicitly do the conversion to floating point. In this case, the SQL statement should explicitly typecast the 1.1 so that DB2 looks for the correct format (the scientific notation format). The **SELECT** query would look like the following:

```
SELECT * FROM TEST WHERE C1 = REAL(1.1)
```

This will give the expected results. The SQL **REAL** function converts the decimal 1.1 to the proper format before DB2 executes the actual select.

Usernames and Passwords Using the ODBC Driver for DB2

When connecting to remote DB2 systems, most users must be authenticated by the remote system by passing a valid user ID and password.

The AS/400 computer is case-sensitive with regard to user ID and password; it accepts them only in uppercase. The ODBC Driver for DB2 automatically converts the user ID and password into uppercase when connecting to a DB2 for OS/400 system.

The mainframe is not case-sensitive; the user ID and password is acceptable in lowercase or uppercase.

DB2 Universal Database (UDB) for Windows NT is case-sensitive; it supports mixed case passwords. Users must enter a password in the correct mixed case. When entering a user ID, use only the Windows NT user name and not the Windows NT domain name.

Errors Returned by the ODBC Driver for DB2

The ODBC Driver for DB2 generates errors in the following areas:

- ODBC Driver Manager
- Microsoft ODBC Driver for DB2
- DRDA Application Requester network client

The ODBC Driver Manager is a shared library that establishes connections with ODBC drivers, submits requests to ODBC drivers, and returns results to applications. An ODBC Driver Manager error has the following format:

```
[vendor] [ODBC DLL] message
```

For example:

```
[Microsoft] [ODBC DLL] Driver does not support this function.
```

If you encounter this type of error, check the last ODBC call the application made for possible problems. For further information on ODBC Driver Manager errors, contact your ODBC application vendor or refer to the ODBC documentation available from Microsoft Press.

An error reported by the Microsoft ODBC Driver for DB2 has the following format:

```
[Microsoft] [ODBC Driver for DB2] message
```

For example:

```
[Microsoft] [ODBC Driver for DB2] Invalid precision specified.
```

If you encounter this type of error is, check the last ODBC call the application made for possible problems. For further information on ODBC Driver errors, contact your ODBC application vendor or refer to the ODBC documentation available from Microsoft Press.

When using the Microsoft ODBC Driver for DB2, data source refers to the target database. An error that occurs in the data source is returned with the data source name and in the following format:

```
[Microsoft] [ODBC Driver for DB2] [data_source] message
```

For example, an ODBC application may receive the following message from a DB2 data source running on an IBM mainframe:

```
[Microsoft] [ODBC Driver for DB2] [DB2] DB2-0919: specified length too long for CHAR column
```

If you encounter this type of error, the application attempted to perform an operation not supported by the DB2 database system. Check the DB2 database system documentation for more information or consult your database administrator.

Two-Phase Commit over TCP/IP Support Using the ODBC Driver for DB2

Two-phase commit (2PC) is a host server-installed protocol that ensures that updates to multiple instances of a database on a network either succeed or fail in their entirety. Host Integration Server 2009 supports 2PC over TCP/IP, allowing you to gain the security of a 2PC connection over the Internet.

Host Integration Server supports 2PC works using three components: the DTC, the Resync Service, and the transaction log. The DTC governs the normal DTC transaction flow: enlist, prepare, commit, and abort. The Resync service coordinates transaction recovery in case of any failure or disconnection, while the transaction log maintains a log of information that is needed in case of recovery.

You can perform a 2PC transaction with the ODBC Driver for DB2 using **SQLSetConnAttr**. Using a 2PC transaction is automatic. However, you may need to configure your 2PC connection using tools such as ODBC Connection Manager.

Troubleshooting the ODBC Driver for DB2

The Windows 2000 Event Viewer can be a useful tool for troubleshooting data access in some cases. The ODBC Driver for DB2 does not issue events. However, when SNA (APPC/LU 6.2) is used for the network transport for the ODBC Driver for DB2, the low-level SNA APPC transport issues events on the SNA connection.

The ODBC Driver for DB2 supplied with Host Integration Server 2009 has the ability to trace DRDA data flows when used over TCP/IP. This capability is accessible from the SNADB2 Service tracing inside the Trace utility shipped with Host Integration Server 2009.

This facility shows the same data as an APPC trace but without the control indicators (for example, What_Received). Socket errors are traced and the error codes can be looked up in Winsock2.h supplied with the Win32 SDK.

When the ODBC Driver for DB2 passes an error code, the best source in which to look-up the meaning of the return code is often the SQL Reference or SQL Messages and Codes Reference for the target SQL database. In this case, the target database is one of the DB2 platforms supported by the ODBC Driver for DB2.

The ODBC Driver for DB2 maintains an internal integer variable named SQLCODE and an internal 5-byte character string variable named SQLSTATE used to check the execution of SQL statements on DB2. SQLCODE is set by DB2 after each SQL statement is executed. DB2 returns the following values for SQLCODE:

- If SQLCODE = 0, execution was successful.
- If SQLCODE > 0, execution was successful with a warning.
- If SQLCODE < 0, execution was not successful.
- SQLCODE = 100, "no data" was found. For example, a **FETCH** statement returned no data because the cursor was positioned after the last row of the result table.

SQLSTATE is also set by DB2 after the execution of each SQL statement. Application programs can check the execution of SQL statements by testing SQLSTATE instead of SQLCODE. SQLSTATE provides application programs with common codes for common error conditions (the values of SQLSTATE are product-specific only if the error or warning is product-specific). Furthermore, SQLSTATE is designed so that application programs can test for specific errors or classes of errors.

SQLSTATE values consist of a two-character class code value, followed by a three-character subclass code value. The first character of an SQLSTATE value indicates whether the SQL statement was executed successfully or unsuccessfully (equal to or not equal to zero, respectively). Class code values represent classes of successful and unsuccessful execution conditions. The following table describes SQLSTATE class codes used by DB2.

Class Code	Description of Error Class
00	Successful completion. Execution of the SQL statement was successful and did not result in any type of warning or exception condition.
01	Warning
02	No data
07	Dynamic SQL error
08	Connection exception
0A	Feature not supported
0F	Invalid token
21	Cardinality violation

22	Data exception
23	Constraint violation
24	Invalid cursor state
25	Invalid transaction state
26	Invalid SQL statement identifier
2D	Invalid transaction termination
34	Invalid cursor name
39	External function call exception
40	Transaction rollback
42	Syntax error or access rule violation
44	WITH CHECK OPTION violation
51	Invalid application state
53	Invalid operand or inconsistent specification
54	SQL or product limit exceeded
55	Object not in prerequisite state
56	Miscellaneous SQL or product error
57	Resource not available or operator intervention
58	System error

The SQLSTATE value of HY000 is defined as a driver-specific error. An SQLSTATE of 08S01 (connection exception with a subclass code of S01) also indicates a driver-specific error. This means the SQLCODE should be looked up in the driver-specific documentation included with the ODBC Driver for DB2.

If the SQLSTATE does not indicate a driver-specific error when the ODBC Driver for DB2 passes back an SQLSTATE of 08S01, it indicates a network error. For example, an SQLCODE of -603 is a driver-specific error that is mapped to DB2OLEDB_COMM_HOST_CONNECT_FAILED in the db2oledb.h include file supplied with the ODBC Driver for DB2. Errors with an SQLSTATE of 08S01 are documented in the db2oledb.h include file (the SQLCODE value) which is located on the Host Integration Server 2009 CD in the SDK\Include subdirectory.

The following steps are useful in researching an error. Start by reading the provided error text returned by the ODBC Driver for DB2. In some cases, the error text may provide limited information. For example, error text from an SQLCODE of -603 reads:

```
Test connection failed because of an error in initializing driver.
Could not connect to specified host.
```

The next step is to lookup the SQLSTATE to determine the source of the error. Is the error a DB2 error, a network client error, or an ODBC Driver error? An SQLSTATE of 08S01 is defined as follows:

```
Communication link failure.
```

This definition is intended to inform the user, administrator, or developer that the error is one related to the ODBC driver's underlying network client.

Unfortunately, many of the SQLSTATE codes returned by the ODBC Driver for DB2 are DB2 errors and are not documented in the ODBC Driver for DB2 Help.

The SQLSTATE of HY000 is defined as a driver-specific error. An SQLSTATE of 08S01 also indicates a driver-specific error. In this case, you should look up the SQLCODE in the driver-specific documentation included with the ODBC Driver for DB2.

If the SQLSTATE does not indicate a driver-specific error, you should look up the SQLCODE in the appropriate DB2 manual for the target platform. For example, an SQLCODE of -603 is documented in Appendix B, "SQLCODEs and SQLSTATEs," in the *AS/400 Advanced Series DB2 for AS/400 SQL Programming, Version 4*, document number SC41-5611-00 published by IBM. An SQLCODE of -603 corresponds to SQLSTATE 23515 in the DB2 for OS/400 error code list. For example, the explanation for this SQLCODE is:

```
Unique index cannot be created because of duplicate keys.
```

When the SQLSTATE and the SQLCODE definitions documented in these appendixes create a mismatch with the actual errors returned, it usually indicates a driver-specific error condition.

A final step in understanding an error is to check the db2oledb.h file. This file is not installed by Setup for the Host Integration Client 2000, but is located on the CD-ROM for in the SDK\Include folder. An SQLCODE (for example, -603) can be found by searching the rightmost column of the db2oledb.h file for a value near to 603. For instance, locate the comment "/* -600 */" and then count down three additional lines to line number 603. The internal error code -603 is defined as follows:

```
DB2OLEDB_COMM_HOST_CONNECT_FAILED.
```

This particular error usually indicates a problem with the configuration parameters or the connection string passed.

ODBC Conformance

The Microsoft ODBC Driver for DB2 supports ODBC 2.*x* and ODBC 3.*x* functions. SQL grammar conformance varies, depending on the version of the DB2 database that is accessed. The following sections list the ODBC functions and attributes supported by the Microsoft ODBC Driver for DB2.

In This Section

Support for ODBC 2 Core Functions

The following table lists the ODBC 2.*x* Core functions that are supported by the Microsoft ODBC Driver for DB2.

ODBC 2.*x* Core Functions	**Functions Supported by the Microsoft ODBC Driver for DB2**
SQLAllocConnect	Yes
SQLAllocEnv	Yes
SQLAllocStmt	Yes
SQLBindCol	Yes
SQLCancel	Yes
SQLColAttributes	Yes
SQLConnect	Yes
SQLDescribeCol	Yes
SQLDisconnect	Yes
SQLError	Yes
SQLExecDirect	Yes
SQLExecute	Yes
SQLFetch	Yes
SQLFreeConnect	Yes
SQLFreeEnv	Yes
SQLFreeStmt	Yes
SQLGetCursorName	Yes
SQLNumResultCols	Yes
SQLPrepare	Yes
SQLRowCount	Yes
SQLSetCursorName	Yes
SQLSetParam	In ODBC 2.0, the ODBC 1.0 **SQLSetparam** function was replaced by **SQLBindParameter**
SQLTransact	Yes

Support for ODBC 2 Level 1 Functions

The following table lists the ODBC 2.*x* level 1 functions that are supported by the Microsoft ODBC Driver for DB2.

ODBC 2.*x* level 1 functions	**Functions supported by the Microsoft ODBC Driver for DB2**
SQLBindParameter	Yes
SQLColumns	Yes
SQLDriverConnect	Yes
SQLGetConnectOption	Yes
SQLGetData	Yes
SQLGetFunctions	Yes
SQLGetInfo	Yes
SQLGetStmtOption	Yes
SQLGetTypeInfo	Yes
SQLParamData	Yes
SQLPutData	Yes
SQLSetConnectOption	Yes
SQLSetStmtOption	Yes
SQLSpecialColumns	Yes
SQLStatistics	Yes
SQLTables	Yes

Support for ODBC 2 Level 2 Functions

The following table lists the ODBC 2.*x* level 2 functions that are supported by the Microsoft ODBC Driver for DB2.

ODBC 2.*x* level 2 functions supported	**Functions supported by the Microsoft ODBC Driver for DB2**
SQLBrowseConnect	No
SQLColumnPrivileges	No
SQLDataSources	Yes. This function is actually supported by the ODBC Driver Manager
SQLDescribeParam	No
SQLDrivers	Yes. This function is actually supported by the ODBC Driver Manager
SQLExtendedFetch	Yes, but supports forward scrolling only
SQLForeignKeys	No
SQLMoreResults	Yes
SQLNativeSQL	Yes
SQLNumParams	Yes
SQLParamOptions	Yes
SQLPrimaryKeys	Yes, but SQL grammar conformance varies depending on the version of the DB2 database that is accessed
SQLProcedureColumns	Yes
SQLProcedures	Yes
SetPos	Yes
SQLSetScrollOptions	Yes
SQLTablePrivileges	No

Support for ODBC 3 Functions

The following table lists the ODBC 3.0 functions that are supported by the Microsoft ODBC Driver for DB2.

ODBC 3.0 functions	Functions supported by the Microsoft ODBC Driver for DB2
SQLAllocHandle	Yes
SQLBulkOperations	No
SQLCloseCursor	Yes
SQLColAttribute	Yes
SQLCopyDesc	Yes
SQLEndTran	Yes
SQLFetchScroll	Yes, but supports forward scrolling only
SQLFreeHandle	Yes
SQLGetConnectAttr	Yes
SQLGetDescField	Yes
SQLDescRec	Yes
SQLGetDiagField	Yes
SQLGetDiagRec	Yes
SQLGetEnvAttr	Yes
SQLGetStmtAttr	Yes
SQLRowCount	Yes
SQLSetConnectAttr	Yes
SQLSetDescField	Yes
SQLSetDescRec	Yes
SQLSetEnvAttr	Yes
SQLSetStmtAttr	Yes

Support for ODBC Connection Attributes

The following table lists the ODBC connection attributes support using the Microsoft ODBC Driver for DB2. Note that the connection attributes in this list use the ODBC Version 3.0 attribute names, rather than the older ODBC 1.0 names.

ODBC connection attribute	ODBC version	Attribute supported by the Microsoft ODBC Driver for DB2
SQL_ATTR_ACCESS_MODE	1.0	Yes
SQL_ATTR_ASYNC_ENABLE	3.0	No
SQL_ATTR_AUTO_IPD	3.0	No
SQL_ATTR_AUTOCOMMIT	1.0	Yes
SQL_ATTR_CONNECTION_DEAD	3.5	No
SQL_ATTR_CONNECTION_TIMEOUT	3.0	No
SQL_ATTR_CURRENT_CATALOG	2.0	Yes
SQL_ATTR_ENLIST_IN_DTC	3.0	Yes
SQL_ATTR_ENLIST_IN_XA	3.0	No
SQL_ATTR_LOGIN_TIMEOUT	1.0	No
SQL_ATTR_METADATA_ID	3.0	No
SQL_ATTR_ODBC_CURSORS	2.0	Yes, handled by ODBC Driver Manager
SQL_ATTR_PACKET_SIZE	2.0	No
SQL_ATTR_QUIET_MODE	2.0	Yes
SQL_ATTR_TRACE	1.0	Yes, handled by ODBC Driver Manager
SQL_ATTR_TRACEFILE	1.0	Yes, handled by ODBC Driver Manager
SQL_ATTR_TRANSLATE_LIB	1.0	No
SQL_ATTR_TRANSLATE_DLL	1.0	No
SQL_ATTR_TRANSLATE_OPTION	1.0	No
SQL_ATTR_TXN_ISOLATION	1.0	Yes

Support for ODBC Statement Attributes

The following table lists the ODBC statement attribute support using the Microsoft ODBC Driver for DB2. Note that the statement attributes in this list use the ODBC Version 3.0 attribute names, rather than the older ODBC 1.0 names.

ODBC statement attribute	ODBC version	Attribute supported by the Microsoft ODBC Driver for DB2
SQL_ATTR_APP_PARAM_DESC	3.0	Yes
SQL_ATTR_APP_ROW_DESC	3.0	Yes
SQL_ATTR_ASYNC_ENABLE	1.0	No
SQL_ATTR_CONCURRENCY		No
SQL_ATTR_CURSOR_SCROLLABLE	3.0	No
SQL_ATTR_CURSOR_SENSITIVITY	3.0	No
SQL_ATTR_CURSOR_TYPE	1.0	Yes, but the ODBC Driver for DB2 supports a forward scrolling only cursor type
SQL_ATTR_ENABLE_AUTO_IPD	3.0	No
SQL_ATTR_FETCH_BOOKMARK_PTR	3.0	No
SQL_ATTR_IMP_PARAM_DESC		Yes, handled by ODBC Driver Manager
SQL_ATTR_IMP_ROW_DESC		Yes, handled by ODBC Driver Manager
SQL_ATTR_KEYSET_SIZE	1.0	No
SQL_ATTR_MAX_LENGTH	1.0	No
SQL_ATTR_MAX_ROWS	1.0	No
SQL_ATTR_METADATA_ID	3.0	Yes
SQL_ATTR_NOSCAN	1.0	Yes
SQL_ATTR_PARAM_BIND_OFFSET_PTR	3.0	Yes
SQL_ATTR_PARAM_BIND_TYPE	3.0	Yes
SQL_ATTR_PARAM_OPERATION_PTR	3.0	Yes
SQL_ATTR_PARAM_STATUS_PTR	3.0	Yes
SQL_ATTR_PARAMS_PROCESSED_PTR	3.0	Yes

SQL_ATTR_PARAMSET_SIZE	3.0	Yes
SQL_ATTR_QUERY_TIMEOUT	1.0	No
SQL_ATTR_RETRIEVE_DATA	1.0	No
SQL_ATTR_ROW_ARRAY_SIZE	3.0	Yes
SQL_ATTR_ROW_BIND_OFFSET_PTR	3.0	Yes
SQL_ATTR_ROW_BIND_TYPE	1.0	Yes
SQL_ATTR_ROW_NUMBER	1.0	No
SQL_ATTR_ROW_OPERATION_PTR	3.0	No
SQL_ATTR_ROW_STATUS_PTR	3.0	Yes
SQL_ATTR_ROWS_FETCHED_PTR	3.0	Yes
SQL_ATTR_SIMULATE_CURSOR	1.0	No
SQL_ATTR_USE_BOOKMARKS	1.0	No

ADO Object Support in the ODBC Driver for DB2

The following table summarizes the Microsoft ActiveX Data Objects (ADO) version 2.0 objects that are supported by the current version of the Microsoft Open Database Connectivity (ODBC) Driver for DB2.

ADO object	Support
Collection	Yes, most methods
Command Object	Yes, some methods, some properties, and all collections
Connection Object	Yes, some methods, some properties, and all collections
Error Object	Yes, some properties
Field Object	Yes, no methods, most properties, and all collections
Parameter Object	Yes, most methods, most properties, and all collections
Recordset Object	Yes, most methods, most properties, and all collections

The **Parameter** object will be supported by a later version of the ODBC Driver for DB2. For more information about ADO object support, see ADO Object Support in the OLE DB Provider for AS/400 and VSAM and ADO Object Support in the OLE DB Provider for DB2.

In This Section

ADO Method Support in the ODBC Driver for DB2

The following table summarizes the ActiveX Data Objects (ADO) version 2.0 object methods that are supported by the current version of the Microsoft ODBC Driver for DB2.

ADO object	Method	Support
Collection Object	**Append** Method	No
	Clear Method	Yes
	Delete Method	Yes
	Item Method	Yes
	Refresh Method	Yes
Command Object	**CreateParameter** Method	Yes
	Cancel Method	No
	Execute Method	Yes
Connection Object	**BeginTrans** Method	Yes
	Cancel Method	No
	Close Method	Yes
	CommitTrans Method	Yes
	Execute Method	Yes
	Open Method	Yes
	OpenSchema Method	Yes
	RollbackTrans Method	Yes
Field Object	**AppendChunk** Method	No
	GetChunk Method	No
	ReadFromFile Method	No
	WriteToFile Method	No
Parameter Object	**AppendChunk** Method	No
Recordset Object	AddNew Method	Yes
	Cancel Method	No
	CancelBatch Method	Yes

	CancelUpdate	Yes
	Clone Method	Yes
	Close Method	Yes
	Delete Method	Yes
	Find Method	No
	GetRows Method	Yes
	Move Method	Yes
	MoveFirst Method	Yes
	MoveLast Method	Yes, when using a client-side cursor only
	MoveNext Method	Yes
	MovePrevious Method	Yes, when using a client-side cursor only
	NextRecordset Method	No
	Open Method	Yes
	Requery Method	Yes
	Resync Method	No
	Save Method	Yes
	Seek Method	No
	Supports Method	Yes
	Update Method	Yes
	UpdateBatch Method	Yes

Note

The **Collection** object is actually a special case, representing a collection of other ADO objects. These collection objects support several methods:

- **Append** to add an object to a collection.
- **Clear** to empty all objects from a collection.
- **Delete** to remove a single object from a collection.
- **Item** to return a specific member object of a collection by name or ordinal number.
- **Refresh** to update the objects in a collection to reflect objects available from and specific to the ODBC Driver.

ADO Property Support in the ODBC Driver for DB2

The following table summarizes the ActiveX Data Objects (ADO) version 2.0 object properties that are supported by the current version of Microsoft ODBC Driver for DB2.

ADO object	Property	Support
Command Object	ActiveConnection Property	Yes
	CommandText Property	Yes
	CommandTimeout Property	No
	CommandType Property	Yes
	Prepared Property	Yes
	State Property	Yes
Connection Object	Attributes Property	Yes
	CommandTimeout Property	No
	ConnectionString Property	Yes
	ConnectionTimeout Property	No
	CursorLocation Property	Yes
	DefaultDatabase Property	No
	IsolationLevel Property	Yes
	Mode Property	Yes
	Provider Property	Yes
	State Property	Yes
	Version Property	Yes
Error Object	Description Property	Yes
	HelpContext Property	No
	HelpFile	No
	NativeError Property	Yes
	Number Property	Yes
	Source Property	Yes
	SQLState Property	Yes

Field Object	ActualSize Property	Yes
	Attributes Property	Yes
	DataFormat Property	No
	DefinedSizeProperty	Yes
	Name Property	Yes
	NumericScale Property	Yes
	OriginalValue Property	Yes
	Precision Property	Yes
	Type Property	Yes
	UnderlyingValue Property	Yes
	Value Property	Yes
Parameter Object	**Attributes** Property	Yes
	Direction Property	Yes
	Name Property	Yes
	NumericScale Property	Yes
	Precision Property	Yes
	Size Property	Yes
	Type Property	Yes
	Value Property	Yes
Recordset Object	**AbsolutePage** Property	No
	AbsolutePosition Property	No
	ActiveCommand Property	Yes
	ActiveConnection Property	Yes
	BOF Property	Yes
	Bookmark Property	Yes
	CacheSize Property	Yes
	CursorLocation Property	Yes

	CursorType Property	Yes
	DataMember Property	No
	DataSource Property	No
	EditMode Property	Yes
	EOF Property	Yes
	Filter Property	No
	Index Property	No
	LockType Property	Yes
	MarshalOptions Property	No
	MaxRecords Property	Yes
	PageCount Property	No
	PageSize Property	No
	RecordCount Property	No
	Sort Property	No
	Source Property	Yes
	State Property	Yes
	Status Property	Yes
	StayInSync Property	No

ADO Collection Support in the ODBC Driver for DB2

The following table summarizes the Microsoft® ActiveX® Data Objects (ADO) version 2.0 object collections that are supported by the current version of the Microsoft ODBC Driver for DB2.

ADO object	Collection	Support
Command Object	**Parameters** Property	Yes
	Properties Property	Yes
Connection Object	**Errors** Property	Yes
	Properties Property	Yes
Field Object	**Properties** Property	Yes
Parameter	**Properties** Property	Yes
Recordset Object	**Fields** Property	Yes
	Properties Property	Yes

Command Object in the ODBC Driver for DB2 (ADO)

The ActiveX Data Objects (ADO) **Command** object is a definition of a specific command that is to be executed against an Open Database Connectivity (ODBC) Driver data source.

Command objects can be used to create a **Recordset** object and obtain records, execute a bulk operation, or manipulate the structure of a database. When using the Microsoft ODBC Driver for DB2, some collections, methods, or properties of a **Command** object can generate an error when called.

The primary purpose of the **Command** object in the context of the ODBC Driver for DB2 is to issue SQL commands for execution by the remote DB2 target server. Legal SQL commands are documented for the target DB2 platforms in SQL Reference Guides published by IBM.

The following table lists the **Command** object methods, properties, and collections that are supported by the current version of the ODBC Driver for DB2.

Name	Comment
Execute Method	Evaluates command text (only supported *Options* parameter for this method is **adCmdText**, which indicates that this is an SQL text command).
ActiveConnection Property	Sets or returns the information used to establish a connection to a data source (see notes following).
CommandText Property	Sets or returns the command text to be executed.
CommandType Property	Sets or returns the type of command in a **CommandText** property.
State Property	Describes the current state of an object.
Properties collection	Collections of properties on the command.

The **Execute** method executes a command and returns a **Recordset** object, if appropriate. The **Command** object can be used to open tables or execute SQL commands on a remote DB2 server. If errors occur, you can examine them with the **Errors** collection on the **Connection** object.

You can create a **Command** object independently of a previously defined **Connection** object by setting the **ActiveConnection** property of the **Command** object to a valid connection string (for the proper syntax, see the **ConnectionString** property of the **Connection** object). ADO still creates a **Connection** object, but it does not assign that object to an object variable. However, if multiple **Command** objects are to be associated with the same connection, the **Connection** object should be explicitly created and opened. This assigns the **Connection** object to an object variable. If the **ActiveConnection** property of the **Command** object is not set to this object variable, ADO creates a new **Connection** object for each **Command** object, even if the same connection string is used.

The **ActiveConnection** property associates an open connection with a **Command** object. The **CommandText** property defines the text version of a command (**SELECT ALL FROM TABLE**, for example). The **CommandType** property specifies the type of command described in the **CommandText** property prior to execution to optimize performance. The **CommandType** property must be set to **adCmdText** for use with the ODBC Driver for DB2.

Connection Object in the ODBC Driver for DB2 (ADO)

The ActiveX Data Objects (ADO) **Connection** object represents an open connection to an Open Database Connectivity (ODBC) data source. The **Provider** property sets the ODBC Driver to use. Setting the **ConnectionString** properties configures the connection before opening the data source. The **Version** property determines the version of the ADO implementation in use.

The **Open** method establishes the physical connection to the data source and the **Close** method terminates the connection. If errors occur, these can be examined with the **Errors** collection.

The following table lists the **Connection** object methods, properties, and collections that are supported by the current version of the Microsoft ODBC Driver for DB2.

Name	Comment
Close Method	Closes a connection to a data source.
Execute Method	Evaluates command text.
Open Method	Opens a connection to a data source and can optionally pass **ConnectionString** parameters with thi s method.
OpenSchema Method	Obtains database schema information from the ODBC Driver.
Attributes Property	One or more characteristics supported for a given **Connection** object.
ConnectionString Prope rty	Contains the information used to establish a connection to a data source (see notes following this tab le).
CursorLocation proper ty	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).
IsolationLevel Property	Sets or returns the level of isolation for a **Connection** object.
Mode Property	Indicates the available permissions for modifying data in a connection.
Provider Property	Sets or returns the name of the provider for a connection.
State Property	Describes the current state of an object.
Version Property	Returns the version number of the ADO implementation in use.
Errors collection	Collections of **Error** objects on the connection.
Properties collection	Collections of properties on the connection.

The information needed to establish a connection to a data source can be set in the **ConnectionString** property or passed as part of the **Open** method. In either case, this information must be in a specific format for use with the ODBC Driver for DB2. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons. The following table list the supported ADO-defined arguments for the **ConnectionString** property.

Argument	Description

Data Source	A required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file that is stored in the Program Files\Common Files\ODBC\Data Sources directory.
File Name	Name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed.
Location	The remote database name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.
Password	Valid mainframe or AS/400 password for use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database.
Provider	Name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "DB2 OLEDB".
User ID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database.

The ODBC Driver for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the following tables. The arguments supported by ODBC Driver for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier ODBC Driver for DB2 included with SNA Server 4.0.

The following table describes the arguments supported by the ODBC Driver for DB2 supplied with Host Integration Server 2009.

Argument	**Description**
BAC	When the *BinAsChar* parameter is set to **true** (1), the ODBC Driver for DB2 treats binary data type fields (with a Code Character Set Identifier (CCSID) of 65535) as character data type fields on a per-data source basis. The **CCSID** and **PCCodePage** values are required input parameters.
CCSID	The CCSID matching the DB2 data as represented on the remote computer. The **CCSID** property is required when processing binary data as character data. Unless the **BinAsChar** value is set, character data is converted based on the DB2 column CCSID and default ANSI code page. If this argument is omitted, this parameter defaults to U.S./Canada (37).
CP	The character *Code Page* to use on the computer. This parameter is required when processing binary data as character data. Unless the Binary as Character (BAC) value is set, character data is converted based on the default ANSI code page configured in Microsoft® Windows®. If this argument is omitted, the default value is set to Latin 1 (1252).
DESC	A field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.

DS	The *Default Schema* parameter is the name of the collection where the ODBC Driver for DB2 looks for catalog information. The *Default Schema* parameter is the "SCHEMA" name for the target collection of tables and views. The ODBC driver uses *Default Schema* to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (for example, ODBC Catalog SQLTables). For DB2, the *Default Schema* is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the *Default Schema* is the target COLLECTION name. For DB2 Universal Database (UDB), the *Default Schema* is the SCHEMA name. If the user does not provide a value for *Default Schema*, then the ODBC driver uses the USER_ID provided at login. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Obviously, this default is inappropriate in many cases, therefore it is essential that the **Default Schema** value in the data source be defined.
DSN	The data source name is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
LLU	When SNA is used for the network transport, this field is the name of the remote logical unit (LU) alias configured in Host Integration Server.
MN	When SNA is used for the Network Transport Library (NTL), the **Mode Name** field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to *QPCSUPP*.
NA	When TCP/IP is used for the NTL, the *Network Address* parameter indicates the IP address or the hostname alias of the host DB2 server.
NP	When TCP/IP is used for the NTL, the *Network Port* parameter indicates the TCP/IP port used for communication with the target DB2 Distributed Relational Database Architecture (DRDA) service. The default value is TCP/IP port 446.
NTL	The *Network Transport Library* parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA. If the default SNA is selected, then values for LLU, MN, and RLU are required. If TCP/IP is selected, then values for **NetAddr** and **NetPort** are required.
PC	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft ODBC Driver for DB2, which is implemented as an IBM DRDA Application Requester, uses packages to issue dynamic and static SQL statements. The ODBC driver will create packages dynamically in the location to which the user points using the *Package Collection* parameter.
PDS	The *Provider Data Source* is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
PROV	Specifies the name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "DB2OLEDB".

PWD	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. Note that this parameter is the same as the *Parameter* parameter.
RDB	The *Remote Database Name* parameter is used as the first part of a three-part, fully qualified DB2 table name. This parameter is referred to by different names depending on the DB2 platform. In DB2 on MVS and OS/390, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 Installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The **RDBNAM** value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no **RDBNAM** value, then a value can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE.
RLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
TPN	The *Transaction Program (TP) Name* parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, the alternative TP Name is set to 0X07F9F9F9.
UID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is the same as the *User ID* parameter.
UOW	Determines whether two-phase commit is enabled. The possible values for this parameter are distributed unit of work (DUW) or remote unit of work (RUW). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** using the ODBC Driver for DB2 is as follows:

```
Conn.Provider="DB2OLEDB"
Conn.ConnectionString = "UID=USERNAME;PWD=password",&_
  "LLU=LOCAL;RLU=DATABASE",&_
  "MN=QPCSUPP;CCSID=37;CP=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic.

When opening a connection object in ADO 2.0, you must specify the *Prompt* connection property. For example, the following

syntax is valid with ADO 1.5 and ADO 2.0 and prompts the user for **ConnectionString** properties, as shown in the following code:

```
Conn.ConnectionString = "Provider=DB2OLEDB
Conn.Properties("PROMPT")=adPromptAlways
Conn.Open
```

A sample **Open** method call with these parameters is as follows:

```
RS.Open "Accounting",Conn,0,1,1
```

The last three parameters to the **Open** method correspond with the *CursorType* (the **adOpenForwardOnly** enum is 0, for example), *LockType* (the **adLockReadOnly** enum is 1, for example), and *Options* (**adCmdText** is 1, which indicates that the source name should be evaluated as SQL text). The *Options* parameter must be set to **adCmdText** (1) when used with a data source name with ODBC Driver for DB2.

The following table describes the allowable values for CCSID when using SNA National Language Support (SNANLS) for character code conversions (the default).

EBCDIC character set	**CCSID value**
Arabic	20420
Binary (No Conversion)	65535
Chinese (Simplified)	935
Chinese (Traditional)	937
Cyrillic (Russian)	20880
Cyrillic (Serbian, Bulgarian)	21025
Denmark/Norway (euro)	1142
Denmark/Norway	20277
Finland/Sweden (euro)	1143
Finland/Sweden	20278
France (euro)	1147
France	20297
Germany (euro)	1141
Germany	20273
Greek (Modern)	875
Greek	20423
Hebrew	20424
Icelandic (euro)	1149

Icelandic	20871
International (euro)	1148
International	500
Italy (euro)	1144
Italy	20280
Japanese (English-lower)	931
Japanese (Extend English)	939
Japanese (Extend Katakana)	930
Japanese (Katakana)	290
Japanese (Katakana-Kanji)	5026
Japanese (Latin-Kanji)	5035
Korean	933
Latin America/Spain (euro)	1145
Latin America/Spain	20284
Latin-1 Open System (euro)	20924
Latin-1 Open System	1047
Multilingual/ROECE (Latin-2)	870
Thai	20838
Turkish (Latin-5)	1026
Turkish	20905
U.S./Canada (euro)	1140
U.S./Canada	37
United Kingdom (euro)	1146
United Kingdom	20285

Note that the SNANLS conversions use the locale configured for the data sources using data links. For more information, see SNA National Language Support Programmer's Guide.

The following table describes the allowable values for CCSID when using ANSI/OEM for character code conversions.

ANSI/OEM character set	**CCSID value**

ANSI - Arabic	1256
ANSI - Baltic	1257
ANSI - Cyrillic	1251
ANSI - Eastern Europe	1250
ANSI - Greek	1253
ANSI - Hebrew	1255
ANSI - Latin I	1252
ANSI - Turkish	1254
ANSI/OEM - Korean (Extended Wansung)	949
ANSI/OEM - Japanese Shift-JIS	932
ANSI/OEM - Simplified Chinese GBK	936
ANSI/OEM - Traditional Chinese Big5	950
ANSI/OEM - Thai	874
ANSI/OEM - Vietnam	1258

Error Object in the ODBC Driver for DB2 (ADO)

The ActiveX Data Objects (ADO) **Error** object contains details about data access errors pertaining to a single operation involving ADO. You can read the properties of an **Error** object to obtain specific details about each error.

The **Error** object does not support any methods or collections; however, the **Errors** collection supported by other objects provides the standard **Collection** methods (**Clear** and **Delete**). **Error** objects are automatically appended to the **Errors** collection by the Open Database Connectivity (ODBC) Driver when they occur. The following table lists the **Error** object properties that are supported by the current version of the Microsoft® ODBC Driver for DB2.

Property Name	**Comment**
Description Property	The text of the error alert that is returned based on the minor error code (specific to the ODBC Driver for DB2) contained in the **Error** object resulting from an error.
NativeError Property	A **Long** integer value of the error code returned by the ODBC Driver for DB2.
Number Property	The **Long** integer value of the ODBC error constant.
Source Property	A **string** that indicates the name of the object or application that originally generated an error.

Field Object in the ODBC Driver for DB2 (ADO)

The ActiveX Data Objects (ADO) **Field** object represents a column of data with a common data type. Each **Field** object corresponds to a column in a **Recordset** object.

The following table lists the **Field** object methods, properties, and collections that are supported by the current version of Microsoft ODBC Driver for DB2.

Name	Comment
ActualSize Property	Actual length of a field's value.
Attributes Property	One or more characteristics supported for a given **Field** object.
DefinedSize Property	Defined size of a **Field** object.
Name Property	Name of the **Field** object.
NumericScale Property	Scale of numeric values in a **Field** object for numeric data.
OriginalValue Property	Value of a **Field** object that existed in the record before changes were made.
Precision Property	Degree of precision for numeric values in a **Field** object for numeric data.
Type Property	Operational type or data type for a **Field** object.
UnderlyingValue Property	Current value of a **Field** object.
Value Property	Value assigned to a **Field** object in a **Recordset**.
Properties collection	Collections of properties on the field.

Recordset Object in the ODBC Driver for DB2 (ADO)

The ActiveX Data Objects (ADO) **Recordset** object represents the entire set of records from a base table. At any time, the **Recordset** object refers to only one record within the set as the current record.

The following table lists the **Recordset** object methods, properties, and collections that are supported by the current version of the Microsoft ODBC Driver for DB2.

Name	Comment
AddNew Method	Creates a new record for an updateable **Recordset** object.
CancelBatch Method	Cancels a pending batch update.
CancelUpdate Method	Cancels any changes made to a current record or to a new record prior to calling the **UpdateBatch** method.
Clone Method	Creates a duplicate **Recordset** object from an existing **Recordset** object.
Close Method	Closes an open object and any dependent objects.
Delete Method	Deletes the current record in an open **Recordset** object or an object from a collection.
GetRows Method	Retrieves multiple records of a **Recordset** into an array.
Move Method	Moves the position of the current record in a **Recordset** object.
MoveFirst Method	Moves to the first record in a specified **Recordset**.
MoveLast Method	Moves to the last record in a specified **Recordset**. This method is only supported when using a client-side cursor.
MoveNext Method	Moves to the next record in a specified **Recordset**.
MovePrevious Method	Moves to the previous record in a specified **Recordset**. This method is only supported when using a client-side cursor.
Open Method	Opens a cursor on a **Recordset**.
Requery Method	Updates the data in a **Recordset** object by re-executing the query on which the object is based (equivalent to calling the **Close** and **Open** methods in succession).
Save Method	Saves a **Recordset** in a file or **Stream** object.
Supports Method	Determines whether a specified **Recordset** object supports a particular type of function.
Update Method	Saves any changes you make to the current record of a **Recordset** object.
UpdateBatch Method	Writes all pending batch updates to disk.
ActiveCommand Property	Returns the **Command** object that created the specified **Recordset**.

ActiveConnection Property	Sets or returns the **Connection** object that the specified **Recordset** object currently belongs.
BOF Property	Indicates whether the current record position is before the first record in a **Recordset** object.
Bookmark Property	Returns a bookmark that uniquely identifies the current record in a **Recordset** object or sets the current record in a **Recordset** object identified by a valid bookmark.
CacheSize Property	Sets or returns the number of records from a **Recordset** object that are cached locally in memory.
CursorLocation Property	Sets or returns the location of the cursor (whether the cursor is on the client or the server side).
CursorType Property	Sets or returns the type of cursor used in a **Recordset** object. Only the **adOpenForwardOnly CursorType** is supported by the current version of the ODBC Driver for DB2.
EditMode Property	Indicates the editing status of the current record type.
EOF Property	Indicates whether the current record position is after the last record in a **Recordset** object.
LockType Property	Sets or returns the types of locks placed on records during editing. The ODBC Driver for DB2 supports locks of type **adLockReadOnly** and **adLockPessimistic**.
MaxRecords Property	Sets or returns the maximum number of records to return to a **Recordset** from a query.
Source Property	Sets or returns the source (table name or command object) for the data in a **Recordset**.
State Property	Describes the current state of an object.
Status Property	Indicates the status of the current record with respect to batch updates or other bulk operations.
Fields collection	Collections of fields on the **Recordset**.
Properties collection	Collections of properties on the **Recordset**.

Managed Provider for DB2 Programmer's Reference

The Managed Provider for DB2 enables users to access IBM DB2 from within a managed code application, and serves as a bridge between a DB2 data source and an ADO DataSet. This section provides the reference material needed to program the Managed Provider for DB2.

Related Sections

Managed Provider Programmer's Guide

Microsoft.HostIntegration.MsDb2Client

Data Access Library Programmer's Reference

The Data Access Library interface enables users to automate lengthy tasks normally performed through the Data Access Tool (DAT) user interface.

Related Sections

Data Access Library Programmer's Guide.

Microsoft.HostIntegration.DataAccessLibrary

Managed Data Provider for Host Files Programmer's Reference

The Managed Data Provider for Host Files enables users to access IBM host file systems from within a managed code application

Related Sections

Managed Data Provider for Host Files Programmer's Guide

SQL Parsing in the Managed Data Provider for Host Files

The following topics describe the parsing rules for the Managed Data Provider for Host Files.

In This Section

SELECT Statement

INSERT Statement

UPDATE Statement

DELETE Statement

Column Collection Parsing

WHERE Clause Parsing

Value Parsing

Parameterized Queries

Limitations

See Also

Concepts

Managed Data Provider for Host Files Programmer's Reference

SELECT Statement

The following describes the **SELECT** statement parsing used in the Managed Provider for Host Files and provides sample **SELECT** statements.

Syntax

SELECT <Column Collection> FROM Filename WHERE <Where Clause>

SELECT <Column Collection> FROM Filename AS <AssemblyName.SchemaName> WHERE <Where Clause>

Examples

```
SELECT * FROM FILE_NAME
SELECT COL_NAME1, COL_NAME2… FROM FILE_NAME
```

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

INSERT Statement

The following describes the syntax of the **INSERT** statement for the Managed Provider for Host Files, and provides sample **INSERT** statements.

Syntax

INSERT INTO Filename VALUES (<Values>)

INSERT INTO Filename AS <AssemblyName.SchemaName> VALUES (<Values>)

INSERT INTO Filename (<Column Collection>) VALUES (<Values>)

INSERT INTO Filename AS <AssemblyName.SchemaName> (<Column Collection>) VALUES (<Values>)

Example

The following are sample **INSERT** statements for the Managed Provider for Host Files.

```
INSERT INTO Filename (COL1, COL2…) VALUES(value1, Value2…)
INSERT INTO Filename AS Library_VSAM.NUMBERS  (OUT1_CHAR1, UNION1, UNION2, UNION3, UNION4,
UNION5, UNION6, OUT1_DECIMAL, OUT1_DOUBLE, OUT1_SINGLE) VALUES  ("RECORD", {1234}, {2765788
53}, {1234}, {271111111}, {-1234}, {-135363175}, 100, 100, 100)Comments
```

See Also

Concepts

Managed Data Provider for Host Files Programmer's Reference

UPDATE Statement

The following describes the **UPDATE** statement for the Managed Provider for Host Files and provides a sample **UPDATE** statement.

Syntax

UPDATE Filename SET (<Values>)

UPDATE Filename SET (<Values>) WHERE <Where Clause>

UPDATE Filename AS <AssemblyName.SchemaName> SET (<Values>)

UPDATE Filename AS <AssemblyName.SchemaName> SET (<Values>) WHERE <Where Clause>

UPDATE Filename SET (<Column Collection>) (<Values>)

UPDATE Filename SET (<Column Collection>) (<Values>) WHERE <Where Clause>

Example

The following is a sample **UPDATE** statement for the Managed Provider for Host Files.

```
UPDATE Filename SET (OUT1_INTEGER) (7) WHERE <Applicable where clause >
```

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

DELETE Statement

The following describes the **DELETE** statement for the Managed Provider for Host files.

Syntax

DELETE FROM Filename

DELETE FROM Filename WHERE <Where Clause>

DELETE FROM Filename AS <AssemblyName.SchemaName>

DELETE FROM Filename AS <AssemblyName.SchemaName> WHERE <Where Clause>

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

Column Collection Parsing

The following describes the column collection parsing for the Managed Provider for Host Files and provides a sample parsed column collection.

1. Check for *** OR**

2. Check for column identifiers separated by commas.

3. After every column name, check whether the **AS** keyword is provided.

   If **AS** is provided, the next token should be the CLR data type, such as **Int32** or **String**).

Example

The following is a sample parsed column collection for the Managed Provider for Host Files.

```
*
Col1, Col2, …
Col1 AS String, Col2 as Int32, …
Col1 AS String, Col2, Col3, …Comments
Optional comments.
```

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

WHERE Clause Parsing

The following describes the parsing rules for a **WHERE** clause for the Managed Provider for Host Files and provides sample parsing statements.

1. Check whether the token is **KEY** or **POSITION**.

2. If the token is **KEY**:

    a. The next token should be an open parenthesis (**(**), followed by the column collection, followed by a close parenthesis (**)**).

    b. The next token can be **=** or **EQUALS** or **BETWEEN**.

    c. If the token is **=** or **EQUALS**, then it should be followed by values. Refer to Value Parsing for details about the values.

    d. If the token is **BETWEEN**, then it should be followed by a value, optionally followed by the **INCLUSIVE** or **EXCLUSIVE** keyword. Then it should be followed by **AND**, followed by another value, optionally followed by **INCLUSIVE** or **EXCLUSIVE**.

3. If the token is **POSITION**:

    a. The next token can be **=** or **EQUALS** or **BETWEEN**.

    b. If the token is **=** or **EQUALS**, then it should be followed by values. Refer to Value Parsing for details about the values.

    c. If the token is **BETWEEN**, then it should be followed by a value, optionally followed by the **INCLUSIVE** or **EXCLUSIVE** keyword. Then it should be followed by **AND**, followed by another value, optionally followed by **INCLUSIVE** or **EXCLUSIVE**.

Example

The following is a sample set of parsing statements for the **WHERE** clause for the Managed Provider for Host Files.

```
WHERE KEY (KEY_COL) = ‘1290’
WHERE KEY (KEY_COL) EQUALS ‘1290’
WHERE KEY (KEY_COL) BETWEEN  1290 AND 1390
WHERE KEY (KEY_COL) BETWEEN  1290 INCLUSIVE AND 1390 EXCLUSIVE
WHERE POSITION = 101
WHERE POSITION EQUALS 101
WHERE POSITION BETWEEN 101 AND 201
WHERE POSITION BETWEEN 101 EXCLUSIVE AND 201 INCLUSIVE
```

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

Value Parsing

The following describes the syntax for value parsing for the Managed Provider for Host Files and provides an example.

Syntax

The primitive values are separated by commas. If there are complex data types like Unions or Arrays, then they are enclosed in curly brackets, like {value}.

Example

The following is an example of value parsing for the Managed Provider for Host Files.

```
("RECORD”,{1234},{276578853},{1234},{271111111},{-1234},
{-135363175},100,100,100)
```

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

Parameterized Queries

The host file provider supports parameterized queries. In this case, instead of using the values directly in the SQL statement, a placeholder can be used.

Syntax

WHERE KEY (KEY_COLUMN) BETWEEN (@p1) AND (@p2)

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

Limitations

The following describes the limitations on SQL statements for the Managed Provider for Host Files.

SQL Limitations

- If the table contains a complex type (for example, Unions), then the **INSERT** and **UPDATE** commands should set the values for all the complex types.
- Data definition language (DDL) commands like **ALTER** and **CREATE** are not supported.
- Any form of data control language (DCL) is not supported.
- The expression: {COLUMN NAME} AS {EXPRESSION} is not supported.

See Also

Other Resources

SQL Parsing in the Managed Data Provider for Host Files

ActiveX Controls Programmer's Reference

Host Integration Server 2009 provides ActiveX controls that enable many Host Integration Server features to be easily accessed from a large number of client development environments. The APIs of these controls are listed in this section.

For general information about programming for ActiveX controls, see the ActiveX Controls Programmer's Guide section of the SDK.

For sample code that uses ActiveX controls, see the Data Integration Samples section in the SDK.

In This Section

Data Queue ActiveX Control Programmer's Reference

This section provides reference information about specific ActiveX methods, properties, and event notifications supported by the Microsoft Data Queue ActiveX control. The function syntax and code examples are based on Microsoft Visual Basic.

In This Section

AddQueueItem Method

The **AddQueueItem** method on an **IEGDataQueue** object adds a record to a specified data queue using the Microsoft® Data Queue ActiveX® control.

Syntax

```
object.AddQueueItem QueueItem, fBlock
```

Parameters

QueueItem

This required parameter representing an **IEIGDataQueueItem** object instance specifying the queue item to add to the queue.

fBlock

This optional parameter specifies whether the operation should block until the completion status is known. This parameter can be set to one of the **eigAnswerYesNoEnum** constants shown in the table following the Parameters section.

Values for the eigAnswerYesNoEnum constants

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the operation should block until the completion status is known.
eigAnswerNo	1	This value indicates that the operation should not block. This is the default value for this parameter if it is omitted from the method call.

Remarks

The queue the item is sent to is represented by **QItemType** property on the **IEIGDataQueueItem**. The *QueueItem*is an object of type **IEIGDataQueueItem** that may have been initialized by the user or returned by a call to **GetQueueItem**.

Cancel Method

The **Cancel** method on an **IEGDataQueue** object terminates a transfer operation that is already in progress. This method cancels an item transfer using the Microsoft® Data Queue ActiveX® control.

Syntax

```
object.Cancel
```

Parameters

None.

Remarks

The **Cancel** method is used to cancel a transfer operation that is already in progress.

CancelQueue Method

The **CancelQueue** method on an **IEGDataQueue** object indicates that an application using the Microsoft® Data Queue ActiveX® control no longer wants to be notified of an incoming queue data item. This can be used to stop pending notifications that were queued as a result of calling **GetQueueItem**.

Syntax

```
object.CancelQueue CancelReqCount, CancelReqType, KeyValue
```

Parameters

CancelReqCount

This optional parameter is a count (a short value) of the number of outstanding queue receive requests to cancel. This parameter defaults to a value of 1.

CancelReqType

This optional parameter specifies the type of queue request to cancel. This parameter can be set to one of the **eigQItemTypeEnum** constants shown in the table following the Parameters section.

KeyValue

This optional parameter is a VARIANT representing the key value and has no default value. If the *CancelReqType* is eigKQItem, then this value specifies the key value to stop waiting for.

Values for the eigQItemTypeEnum constants

Enumeration	Value	Description
eigKQItem	0	A keyed item.
eigQItem	1	A non-keyed item. This is the default value for this parameter if it is not specified.

This parameter defaults to **eigQItem**.

Remarks

An application can call the **CancelQueue** method passing a *CancelReqCount* value of 0, in order to retrieve the number of queued receive requests.

It is possible that a notification event is in process during this method call. If this occurs, the application may receive a notification following the successful completion of this function. Entering a *CancelReqCount* value that is larger than the queued requests will result in all requests being cancelled. No error will be returned under this condition. Calling this method while no queued requests are outstanding will result in a noncritical error return code indicating this condition.

CCSID Property

The **CCSID** property on an **IEGDataQueueCtrl** object indicates the character code set identifier (CCSID) that must match the data in the queue as represented on the remote host computer. This property affects how data conversion is handled using the Microsoft® Data Queue ActiveX® control. This property sets or returns a short value representing a host CCSID for the data file. The default value for this property is 37 representing a CCSID for U.S./Canada.

Syntax

```
current CCSID = object.CCSID
object.CCSID = 37
```

Remarks

The **CCSID** property is used to set or return the character code set identifier (CCSID) matching the data in the data queue as represented on the remote host computer. This value is used for data conversion of any character data in the host data queue to the **PCCodePage** property specified representing ANSI or Unicode character data on the computer running Microsoft® Windows®.

ClearAll Method

The **ClearAll** method on the **EIGDataQueue** object removes all items from the queue.

Syntax

```
object.ClearAll OverWrite
```

Parameters

OverWrite

This optional parameter specifies whether to overwrite data in the queue. This parameter can be set to one of the **eigAnswerYesNoEnum** constants shown in the table following the Parameters section.

Values for the eigAnswerYesNoEnum constants

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the operation should overwrite data in the queue. This is the default value for this parameter if it is omitted from the method call.
eigAnswerNo	1	This value indicates that the operation should not overwrite data in the queue.

Connect Method

The **Connect** method on an **IEGDataQueueCtrl** object establishes a connection to the configured host using the Microsoft® Data Queue ActiveX® control and reports to the user an indication of the success or failure of the action.

Syntax

```
object.Connect
```

Parameters

None.

Remarks

The **Connect** method is used to establish a connection to the host.

If the Data Queue ActiveX control support DLL cannot be loaded, the Win32® **SetErrorMode** function is called with an error value of SEM_NOOPENFILEERRORBOX | SEM_FAILCRITICALERRORS. Other types of errors are returned using the **ISupportErrorInfo** object.

ConnectionState Property

The **ConnectionState** property on an **IEGDataQueueCtrl** object indicates the current state of the connection to the host using the Microsoft® Data Queue ActiveX® control. The state of a connection can be unspecified, idle, connecting, connected, or disconnecting. This property returns a Long value representing a **eigConnectionStateEnum**. The default value for this property is **eigConnStateIdle**.

Syntax

```
currentConnectionState = object.ConnectionState
```

Remarks

The **ConnectionState** property is used to return the current state of the connection to the host. This property is read-only and can be one of the following **eigConnectionStateEnum** constants:

Enumeration	**Value**	**Description**
eigConnStateUnspecified	-1	This value indicates that the connection state is unspecified.
eigConnStateIdle	0	This value indicates that the connection state is idle and no connection to the host exists.
eigConnStateConnecting	1	This value indicates that the connection state is in the process of connecting to the host.
eigConnStateConnected	2	This value indicates that the connection state is connected indicating a connection to the host.
eigConnStateDisconnecting	3	This value indicates that the connection state is in the process of disconnecting from the host.

ConnectionType Property

The **ConnectionType** property on an **IEGDataQueueCtrl** object indicates the network transport used for this connection. This property sets or returns a Long value representing an **eigConnectionTypeEnum**. The default value for this property is **eigConnTypeAPPC** indicating an APPC connection using SNA.

Syntax

```
currentConnectionType = object.ConnectionType
```

Remarks

The **ConnectionType** property is used to set or return the connection type used to connect to the host. This property can be one of the following **eigConnectionTypeEnum** constants:

Enumeration	Value	Description
eigConnTypeUnspecified	-1	This value indicates that the connection type is unspecified.
eigConnTypeAPPC	0	This value indicates an APPC connection to the host using SNA LU 6.2.

If APPC (SNA LU 6.2) is selected for **ConnectionType**, then values for the **LocalLU**, **ModeName**, and **RemoteLU** properties are required.

CreateQueue Method

The **CreateQueue** method on an **IEGDataQueue** object creates a data queue using the Microsoft® Data Queue ActiveX® control. Following the successful creation of the queue, the object has a virtual connection to the data queue, such that a single instance of this object represents a single queue connection. In order to break this connection, the application can modify the **QueueName** property thus altering the queue association to a new value.

Syntax

```
object.CreateQueueMaxMsgLength, QAuthority, QueueClass,
        AddSenderInfo, HostCCSID, InitialSize, queueLoc, recordLenCls,
        Title, AllowDupKeys, MakeKeyLen
```

Parameters

MaxMsgLength

This required parameter indicates the maximum length of a record in the queue represented as a short integer with possible values ranging from 1 to 31,744. This parameter defaults to a value of 256.

QAuthority

This required parameter specifies the authority to grant users of this queue. This parameter can be set to one of the **eigQAuthorityEnum** constants shown in the table following the Parameters section.

QueueClass

This required parameter indicates how the data will be received from the data queue. This parameter can be set to one of the **eigQClassEnum** constants shown in the table following the Parameters section.

AddSenderInfo

This required parameter indicates whether the queue sender's ID should be kept. This parameter can be set to one of the **eigAnswerYesNoEnum** constants shown in the table following the Parameters section.

HostCCSID

This optional parameter indicates the character code set identifier (CCSID) represented as a short to be used on the host and affects how data conversion is handled. This value must match the data in the queue as represented on the remote host computer. This parameter has no default value, but if this parameter is not specified the *HostCCSID* defaults to the value of the **CCSID** property set on the **IEGDataQueueCtl** object.

InitialSize

This optional parameter indicates the initial data queue size represented as a short integer.

queueLoc

This optional parameter indicates the queue location represented as VARIANT_BOOL.

recordLenCls

This optional parameter indicates the record length class. This parameter can be set to one of the **eigRecordLenClsEnum** constants shown in the table following the Parameters section.

Title

This optional parameter indicates a text description of the queue represented as a BSTR with a maximum length of 50 characters. The default value for this parameter is an empty string.

AllowDupKeys

This optional parameter indicates whether duplicate keys are allowed. This parameter can be set to one of the **eigAnswerYesNoEnum** constants shown in the table following the Parameters section.

Note that this parameter is only valid if the *QueueClass* is specified as **eigQClassKeyed**.

MakeKeyLen

This optional parameter indicates the maximum length of a key for this data queue represented as a short integer ranging

from 1 to 256. Note that this parameter is only valid if the *QueueClass* is specified as **eigQClassKeyed**. This parameter has no default value.

The eigQAuthorityEnum constants

Enumeration	Value	Description
eigQAuthUnspecified	-1	This value indicates that the authority is unspecified.
eigQAuthDefault	0	This value indicates directory default authorization. This is the default value for this parameter if it is not specified.
eigQAuthAll	1	This value indicates all authorization.
eigQAuthExclude	2	This value indicates exclude authorization.
eigQAuthChange	3	This value indicates change authorization.
eigQAuthUse	4	This value indicates use authorization.
eigQAuthLibCreate	5	This value indicates library create authorization.

The eigQClassEnum constants

Enumeration	Value	Description
eigQClassUnspecified	-1	No value
eigQClassFIFO	0	A first in, first out queue. This is the default value for this parameter if it is not specified.
eigQClassLIFO	1	A last in, first out queue.
eigQClassKeyed	2	A keyed ordered queue.

The eigAnswerYesNoEnum constants

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the sender ID should be kept.
eigAnswerNo	1	This value indicates that the sender ID should not be kept. This is the default value for this parameter if it is not specified in the method call.

The eigRecordLenClsEnum constants

Enumeration	Value	Description
eigRecordLenUnspecified	-1	This value indicates that the record length class is unspecified.
eigRecordLenFixed	0	This value indicates fixed length records. This is the default value for this parameter if it is not specified.
eigRecordLenInitVarLen	1	This value indicates initial variable record length.
eigRecordLenVarLen	2	This value indicates variable record length.

Remarks

The type of data queue created is dependent on the value of the *QueueClass* parameter. If a *QueueClass* of **eigQClassKeyed** is specified, then a keyed data queue is created. A *QueueClass* of **eigQClassFIFO** and **eigQClassLIFO** will result in a non-keyed data queue being created.

CreateQueueContainer Method

The **CreateQueue** method on an **IEGDataQueueCtl** object creates an instance of an **IEIGDataQueue** container object using the Microsoft® Data Queue ActiveX® control and optionally initializes the **QueueName** property. The default value for this property is VT_EMPTY.

Syntax

```
object.CreateQueueContainer QueueName
```

Parameters

QueueName

This optional parameter is a BSTR string representing the name of the data queue that this object instance is connected to. This parameter corresponds with the value for the QueueName property.

Remarks

The created queue object is assumed to be associated with the connection object that created it for the life of the connection or the life of the queue object.

DeleteQueue Method

The **DeleteQueue** method on an **IEGDataQueue** object clears all messages from the queue and then deletes the queue using the Data Queue ActiveX® control.

Syntax

```
object.DeleteQueue OverWriteData
```

Parameters

OverWriteData

This required parameter indicates whether data should be overwritten in the data queue. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

The eigAnswerYesNoEnum constants

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the operation should overwrite data in the queue. This is the default value for this parameter if it is omitted from the method call.
eigAnswerNo	1	This value indicates that the operation should not overwrite data in the queue.

This parameter had a default value of **eigAnswerYes.**

Disconnect Method

The **Disconnect** method on an **IEGDataQueueCtrl** object terminates an existing connection to a host computer using the Microsoft® Data Queue ActiveX® control.

Syntax

```
object.Disconnect
```

Parameters

None.

Remarks

The **Disconnect** method is used to terminate a connection to the host. Errors are returned using the **ISupportErrorInfo** object.

GetQueueItem Method

The **GetQueueItem** method on an **IEGDataQueue** object receives an item from a keyed queue using the Microsoft® Data Queue ActiveX® control.

Syntax

```
object.GetQueueItemQueueType, BlockComplete, PeekQItem,
        ProvideExtInfo, TimeOut, UserProfile, SenderInfo, SearchKey,
        SearchOrder, QueueItem
```

Parameters

QueueType

This required parameter specifies the type of the queue request. This parameter can be set to one of the **eigQItemTypeEnum** constants listed in the table following the Parameters section.

BlockComplete

This required parameter specifies whether the operation should block until the completion status is known. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

PeekQItem

This required parameter indicates whether to keep the record in the data queue. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

ProvideExtInfo

This required parameter indicates whether to provide information in the External Job, name, and user properties. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

TimeOut

This required parameter indicates the amount of time in seconds represented as a short value to block before indicating a failure. This parameter has a default value of 0.

UserProfile

This required parameter indicates whether to provide user profile feedback. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

SenderInfo

This required parameter indicates whether the sender's ID should be returned. This parameter can be set to one of the **eigAnswerYesNoEnum** constants listed in the table following the Parameters section.

SearchKey

This optional parameter indicates the key used in conjunction with the *SearchOrder* parameter used to identify the queue data item being requested. This must be the same length as specified in the **CreateDataQueue** method call. This parameter is represented as a VARIANT and has no default value.

Note that this parameter is only valid when the *QueueClass* for the data queue is **eigQClassKeyed**.

SearchOrder

This optional parameter indicates the relational order used in receiving keyed data queue items. This parameter can be set to one of the **eigSearchKeyEnum** constants listed in the table following the Parameters section.

Note that this parameter is only valid when the *QueueClass* for the data queue is **eigQClassKeyed**.

QueueItem

This returned parameter is the requested **IEGDataQueueItem**.

The eigQItemTypeEnum constants

Enumeration	Value	Description

eigKQItem	0	A keyed item.
eigQItem	1	A non-keyed item. This is the default value for this parameter if it is not specified.

The eigAnswerYesNoEnum constants

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the sender ID should be returned.
eigAnswerNo	1	This value indicates that the sender ID should not be returned. This is the default value for this parameter if it is not specified in the method call.

The eigSearchKeyEnum constants

Enumeration	Value	Description
eigSearchKeyUnspecified	-1	This value indicates that the *SearchOrder* parameter is unspecified.
eigSearchKeyEqual	0	This value indicates a search for items equal to the specified *SearchKey* parameter.
eigSearchKeyGreaterThan	1	This value indicates a search for items greater than the specified *SearchKey* parameter.
eigSearchKeyLessThan	2	This value indicates a search for items less than the specified *SearchKey* parameter.
eigSearchKeyGreaterEqual	3	This value indicates a search for items greater than or equal to the specified *SearchKey* parameter.
eigSearchKeyLessEqual	4	This value indicates a search for items less than or equal to the specified *SearchKey* parameter.

Remarks

If the *BlockComplete* parameter is **eigAnswerNo** and no queue item is available, the request will be queued. Following the receipt of a queue data item, an event will be fired to the client indicating the availability of data. The client application will be required to call this function again in order to receive the queue data.

If the *BlockComplete* parameter is **eigAnswerYes** and no data arrives on the queue within the specified amount of time, the operation is cancelled and no data is returned. An error indication is returned to the client indicating that a timeout has occurred.

Each call to this method may result in a queued process. Multiple calls may result in multiple notifications.

If the *SenderInfo* item is empty, then the information will not be returned. The client must allocate the storage as an indication that it wishes to receive this information. The type of data queue is dependent on the value of the *QueueClass* parameter when the data queue is created. For a *QueueClass* of **eigQClassKeyed**, a keyed data queue is created. A *QueueClass* of **eigQClassFIFO** or **eigQClassLIFO** will result in a non-keyed data queue being created.

LocalLU Property

The **LocalLU** property on an **IEGDataQueueCtrl** object indicates the local LU alias for an APPC (SNA LU 6.2) connection type to the remote host computer using the Microsoft® Data Queue ActiveX® control. This property sets or returns a BSTR string value representing the local LU name. The default value for this property is the "LOCAL" string.

Syntax

```
currentLocalLu = object.LocalLU
object.LocalLu = "Local2"
```

Remarks

The **LocalLU** property is used to set or return the local LU alias. When LU 6.2 (SNA) is selected for the ConnectionType property, this property must match the name of the local LU alias configured using SNA Manager.

ModeName Property

The **ModeName** property on an **IEGDataQueueCtrl** object indicates the APPC mode used for an APPC (SNA LU 6.2) connection type to the remote host computer using the Microsoft® Data Queue ActiveX® control. This property sets or returns a BSTR string value representing the APPC mode. The default value for this property is the "QPCSUPP" string.

Syntax

```
currentAppcMode = object.ModeName
object.ModeName = "QPCSUPP"
```

Remarks

The **ModeName** property is used to set or return the APPC mode. When APPC (LU 6.2 SNA) is selected for the ConnectionType property, this field must bet set to the APPC mode that matches the host configuration and Microsoft® Host Integration Server configuration.

Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security).

Password Property

The **Password** property on an **IEGDataQueueCtrl** object indicates the password used for authentication. This property affects connection and authentication to a host computer using the Microsoft® Data Queue ActiveX® control. This property sets or returns a BSTR and has no default value.

Syntax

```
currentUserPassword = object.Password
object.Password = Dialog.UserPassword
```

Remarks

The **Password** property is used to set or return the password used for authenticating the user on a host computer. A valid user name and password are normally required to access data on a host computer. The password is case-sensitive and is normally displayed as asterisks in a dialog box for security purposes.

PCCodePage Property

The **PCCodePage** property on an **IEGDataQueueCtrl** object indicates the code page to be used on the PC. This property affects how data conversion is handled using the Microsoft® Data Queue ActiveX® control. This property sets or returns a short value representing the PC code page for the data file. The default value for this property is 1252 representing a PC code page of Latin 1.

Syntax

```
currentPCCodePage = object.PCCodePage
object.PCCodePage = 1252
```

Remarks

The **PCCodePage** property is used to set or return the code page to be used on the PC. This value is used for data conversion of any character data in the host file to ANSI or Unicode character data in the local file on the computer running Microsoft® Windows®.

QueryAttribute Method

The **QueryAttribute** method on the **EIGDataQueue** object returns the value of an attribute associated with a data queue.

Syntax

```
object.QueryAttribute Attribute, Value
```

Parameters

Attribute

This required parameter indicates the attribute value to be retrieved. This parameter can be set to one of the **eigAttributeEnum** constants listed in the table following the Parameters section.

Value

This required parameter points to a variant that will receive the value for this attribute.

The eigAttributeEnum constants

Enumeration	Value	Description
eigAttributeUnspecified	-1	No value
eigAttributeCCSID	0	The code character set identifier used on the host.
eigAttributeDirName	1	Directory name
eigAttributeDataClass	2	Data class name
eigAttributeKeyDef	3	Key definition
eigAttributeDupKeys	4	Indicates whether the duplicate keys capability is enabled for this data queue.
eigAttributeMgmCls	5	Management class name
eigAttributeQueCls	6	The queue class corresponding with the *QueueClass* parameter on the **CreateQueue** method.
eigAttributeSize	7	The queue initial size
eigAttributeQLoc	8	Queue location
eigAttributeSenderInfo	9	Queue senders ID kept
eigAttributeMaxMsgLen	10	The maximum record length for a message.
eigAttributeRecLenClass	11	Record length class
eigAttributeStgClass	12	Storage class name
eigAttributeTitle	13	Description text

Remarks

Most of these attributes correspond with the parameters specified to the CreateQueue method when the data queue is created.

QueueName Property

The **QueueName** property on an **IEGDataQueue** object indicates the name of the data queue this object is associated with.

Syntax

```
currentQueueName = object.QueueName
object.QueueName = "OrderEntry"
```

Remarks

The value of the **QueueName** property is the name of the queue represented as a BSTR string.

Following the successful creation of a queue using the CreateQueue method, the object has a virtual connection to the data queue, such that a single instance of this object represents a single queue connection. In order to break this connection, a client application can modify the **QueueName** property thus altering the queue association to a new value.

RemoteLU Property

The **RemoteLU** property on an **IEGDataQueueCtrl** object indicates the remote LU alias for an APPC (SNA) connection type to the remote host computer using the Microsoft® Data Queue ActiveX® control. This property sets or returns a BSTR string value representing the remote LU name. This property has no default value.

Syntax

```
currentRemoteLu = object.RemoteLU
object.RemoteLu = "Remote10"
```

Remarks

The **RemoteLU** property is used to set or return the remote LU alias. When LU 6.2 (SNA) is selected for the ConnectionType property, this property must match the name of the remote LU alias configured using SNA Manager.

SetAttribute Method

The **SetAttribute** method on the **EIGDataQueue** object changes an attribute associated with a data queue.

Syntax

```
object.SetAttribute Attribute, Value
```

Parameters

Attribute

This required parameter indicates the attribute to be set. This parameter can be set to one of the **eigAttributeEnum** constants listed in the table following the Parameters section.

Value

This required parameter specifies a variant representing the value to set for this attribute.

The eigAttributeEnum constants

Enumeration	Value	Description
eigAttributeUnspecified	-1	No value
eigAttributeCCSID	0	Coded character set identifier
eigAttributeDirName	1	Directory name
eigAttributeDataClass	2	Data class name
eigAttributeKeyDef	3	Key definition
eigAttributeDupKeys	4	Duplicate keys capability
eigAttributeMgmCls	5	Management class name
eigAttributeQueCls	6	The queue class corresponding with the *QueueClass* parameter on the **CreateQueue** method.
eigAttributeSize	7	Queue initial size
eigAttributeQLoc	8	Queue location
eigAttributeSenderInfo	9	Queue senders ID kept
eigAttributeMaxMsgLen	10	Record length
eigAttributeRecLenClass	11	Record length class
eigAttributeStgClass	12	Storage class name
eigAttributeTitle	13	Description text

Remarks

Most of these attributes correspond with the parameters specified to the CreateQueue method when the data queue is created.

StopQueue Method

The **StopQueue** method on an **IEGDataQueue** object suspends send and receive operations for a queue using the Microsoft® Data Queue ActiveX® control.

Syntax

```
object.StopQueue
```

Parameters

None.

UserID Property

The **UserID** property on an **IEGDataQueueCtrl** object indicates the user name used for authentication. This property affects connection and authentication to a host computer using the Microsoft® Data Queue ActiveX® control. This property sets or returns a BSTR and has no default value.

Syntax

```
currentUserName = object.UserID
object.UserID = Dialog.UserName
```

Remarks

The **UserID** property is used to set or return the user name used for authenticating the user on a host computer. A valid user name and password are normally required to access data on a host computer. The user name is case-sensitive.

Host File Transfer ActiveX Control Programmer's Reference

This section provides reference information about specific ActiveX methods, properties, and event notifications supported by the Microsoft Host File Transfer ActiveX control. The function syntax and code examples are based on Microsoft Visual Basic.

In This Section

AppendToEnd Property

The **AppendToEnd** property on an **IEIGFileTransferCtl** object indicates whether a file transfer operation should append to the end of a file if the file exists, or whether a file transfer operation should overwrite the existing contents replacing the data with the new information. This property affects file transfer operations using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a Long value representing an **eigAnswerYesNoEnum**. The default value for this property is **eigAnswerYes**.

Syntax

```
currentAppendFlag = object.AppendToEnd
object.AppendToEnd = eigAnswerYes
```

Remarks

The **AppendToEnd** property is used to set or return a flag that indicates whether a file transfer operation will append to the end of a file or overwrite the file. This property can be set to one of the following **eigAnswerYesNoEnum** constants:

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the file transfer operation will append to the end of a file if it exists.
eigAnswerNo	1	This value indicates that the file transfer operation will overwrite the existing contents of a file if it exists.

The **AppendToEnd** property and the **OverwriteHostFile** property are mutually exclusive, so it is not possible to enable (set to yes) one of these properties before the opposing property is disabled (set to no). The **AppendToEnd** property takes precedence over the **OverwriteHostFile** property, since **AppendToEnd** defaults to yes and **OverwriteHostFile** defaults to no. Consequently, the order in which these properties are set will affect the outcome. For example, the following order will result in the properties being set correctly:

```
FileTransfer.AppendToEnd = eigAnswerNo         // correctly set to no
FileTransfer.OverwriteHostFile = eigAnswerYes  // correctly set to yes
```

In contrast, setting the properties in the improper order will cause the properties to be set incorrectly as follows:

```
FileTransfer.OverwriteHostFile = eigAnswerYes   // remains at no
// AppendToEnd defaults to eigAnswerYes, so this change is illegal
FileTransfer.AppendToEnd = eigAnswerNo          // correctly set to no
```

In this second case, the **OverwriteHostFile** property cannot be set to yes (enabled) until the **AppendToEnd** property is set to no (disabled).

Cancel Method

The **Cancel** method on an **IEIGFileTransferCtl** object terminates a file transfer operation that is already in progress. This method cancels a file transfer using the Microsoft® Host File Transfer ActiveX® control.

Syntax

```
object.Cancel
```

Parameters

None.

Remarks

The **Cancel** method is used to cancel a file transfer operation that is already in progress. If the **Cancel** method is executed while uploading a file with the **AppendToEnd** property set to yes, this will result in no change to the host file. However, if the **Cancel** method is executed while uploading a file with the **OverwriteHostFile** property set to yes, this will result in an empty host file. The **Cancel** method implies the transfer has been stopped and all the files are at their original values, but this is not really the case when the **OverwriteHostFile** property is set to yes.

CCSID Property

The **CCSID** property on an **IEIGFileTransferCtl** object indicates the character code set identifier (CCSID) that must match the data in the file as represented on the remote host computer. This property affects how data conversion is handled using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a short value representing a host CCSID for the data file. The default value for this property is 37 representing a CCSID for U.S./Canada.

Syntax

```
current CCSID = object.CCSID
object.CCSID = 37
```

Remarks

The **CCSID** property is used to set or return the character code set identifier (CCSID) matching the data in the file as represented on the remote host computer. This value is used for data conversion of any character data in the host file to the **PCCodePage** property specified representing ANSI or Unicode character data on the Microsoft® Windows® computer.

Connect Method

The **Connect** method on an **IEIGFileTransferCtl** object establishes a connection to the configured host using the Microsoft® Host File Transfer ActiveX® control and reports to the user an indication of the success or failure of the action.

Syntax

```
object.Connect
```

Parameters

None.

Remarks

The **Connect** method is used to establish a connection to the host.

If the Host File Transfer ActiveX control support DLL cannot be loaded, the Win32® **SetErrorMode** function is called with an error value of SEM_NOOPENFILEERRORBOX | SEM_FAILCRITICALERRORS. Other types of errors are returned using the **ISupportErrorInfo** object.

ConnectionState Property

The **ConnectionState** property on an **IEIGFileTransferCtl** object indicates the current state of the connection to the host using the Microsoft® Host File Transfer ActiveX® control. The state of a connection can be unspecified, idle, connecting, connected, or disconnecting. This property returns a Long value representing an **eigConnectionStateEnum**. The default value for this property is **eigConnStateIdle**.

Syntax

```
currentConnectionState = object.ConnectionState
```

Remarks

The **ConnectionState** property is used to return the current state of the connection to the host. This property is read-only and can be one of the following **eigConnectionStateEnum** constants:

Enumeration	Value	Description
eigConnStateUnspecified	-1	This value indicates that the connection state is unspecified.
eigConnStateIdle	0	This value indicates that the connection state is idle and no connection to the host exists.
eigConnStateConnecting	1	This value indicates that the connection state is in the process of connecting to the host.
eigConnStateConnected	2	This value indicates that the connection state is connected indicating a connection to the host.
eigConnStateDisconnecting	3	This value indicates that the connection state is in the process of disconnecting from the host.

ConnectionType Property

The **ConnectionType** property on an **IEIGFileTransferCtl** object indicates the network transport used for this connection. The **ConnectioneType** property designates whether the Microsoft® Host File Transfer ActiveX® control connects via APPC (SNA LU 6.2) or TCP/IP. This property sets or returns a Long value representing an **eigConnectionTypeEnum**. The default value for this property is **eigConnTypeAPPC** indicating an APPC connection using SNA.

Syntax

```
currentConnectionType = object.ConnectionType
object.ConnectionType = eigConnTypeTCPIP
```

Remarks

The **ConnectionType** property is used to set or return the connection type used to connect to the host. This property can be one of the following **eigConnectionTypeEnum** constants:

Enumeration	Value	Description
eigConnTypeUnspecified	-1	This value indicates that the connection type is unspecified.
eigConnTypeAPPC	0	This value indicates an APPC connection to the host using SNA LU 6.2.
eigConnTypeTCPIP	1	This value indicates a TCP/IP connection to the host.

If APPC (SNA) is selected for **ConnectionType**, then values for the **LocalLU**, **ModeName**, and **RemoteLU** properties are required.

If TCP/IP is selected for **ConnectionType**, then values for the **NetAddr** and **NetPort** properties are required.

CreateIfNonExisting Property

The **CreateIfNonExisting** property on a **IEIGFileTransferCtl** object indicates whether a file transfer operation should create a new destination file if one does not already exist. This property affects file transfer operations using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a Long value representing an **eigAnswerYesNoEnum**. The default value for this property is **eigAnswerNo**.

Syntax

```
currentCreateFlag = object.CreateIfNonExisting
object.CreateIfNonExisting = eigAnswerYes
```

Remarks

The **CreateIfNonExisting** property is used to set or return a flag that indicates whether a file transfer operation should create a new destination file if one does not already exist. This property can be set to one of the following **eigAnswerYesNoEnum** constants:

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the file transfer operation will create a new destination file if the file does not already exist.
eigAnswerNo	1	This value indicates that the file transfer operation will not create a new destination file if the file does not already exist. If the destination file does not already exist, then the file copy operation will not take place.

Disconnect Method

The **Disconnect** method on an **IEIGFileTransferCtl** object terminates an existing connection to a host computer using the Microsoft® Host File Transfer ActiveX® control.

Syntax

```
object.Disconnect
```

Parameters

None.

Remarks

The **Disconnect** method is used to terminate a connection to the host. Errors are returned using the **ISupportErrorInfo** object.

GetFile Method

The **GetFile** method on an **IEIGFileTransferCtl** object copies a file from host storage to local storage using the Microsoft® Host File Transfer ActiveX® control.

Syntax

```
object.GetFile LocalFile, HostFile
```

Parameters

LocalFile

This required parameter specifies the path of a local file that will be written to as a result of this operation.

HostFile

This required parameter specifies the name of the host file that will be copied to the local file.

Remarks

The **GetFile** method can only be called after a connection has been established to the host (when the ConnectionState property is connected). The behavior of the **GetFile** method is affected by the values of the AppendToEnd, CreateIfNonExisting, and OverwriteHostFile properties.

Errors are returned for this method using the **ISupportErrorInfo** object.

LocalLU Property

The **LocalLU** property on an **IEIGFileTransferCtl** object indicates the local LU alias for an APPC (SNA) connection type to the remote host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value representing the local LU name. The default value for this property is the "LOCAL" string.

Syntax

```
currentLocalLu = object.LocalLU
object.LocalLu = "Local2"
```

Remarks

The **LocalLU** property is used to set or return the local LU alias. When LU 6.2 (SNA) is selected for the ConnectionType property, this property must match the name of the local LU alias configured using SNA Manager.

This property is ignored when TCP/IP is selected for the **ConnectionType** property.

ModeName Property

The **ModeName** property on an **IEIGFileTransferCtl** object indicates the APPC mode used for an APPC (SNA) connection type to the remote host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value representing the APPC mode. The default value for this property is the "QPCSUPP" string.

Syntax

```
currentAppcMode = object.ModeName
object.ModeName = "QPCSUPP"
```

Remarks

The **ModeName** property is used to set or return the APPC mode. When APPC (LU 6.2 SNA) is selected for the ConnectionType property, this field must bet set to the APPC mode that matches the host configuration and Host Integration Server configuration. This property is ignored when TCP/IP is selected for the **ConnectionType** property.

Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bi-directional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security).

NetAddr Property

The **NetAddr** property on an **IEIGFileTransferCtl** object indicates the IP address of the host computer for a TCP/IP connection type to the remote host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value representing the IP address of the host computer. This property has no default value.

Syntax

```
currentHostIP = object.NetAddr
object.NetAddr = "207.136.131.30"
```

Remarks

The **NetAddr** property is used to set or return the IP address of the host computer. When TCP/IP is selected for the ConnectionType property, this property must match the IP address of the host computer used where files will be transferred. This property can be an IP address or the name representing the host IP address using the Domain Name System (sna.microsoft.com, for example). This property is ignored when SNA is selected for the **ConnectionType** property.

NetPort Property

The **NetPort** property on an **IEIGFileTransferCtl** object indicates the TCP/IP port used for communication with the host for a TCP/IP connection type to the remote host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value representing the TCP/IP port used for communication with the host. The default value for this property is the string "446" representing TCP/IP port 446.

Syntax

```
currentIPPort = object.NetPort
object.NetPort = "446"
```

Remarks

The **NetPort** property is used to set or return the TCP/IP port used for communication with the host. When TCP/IP has been selected for the ConnectionType property, this parameter is the TCP/IP port used for communication with the host. This property is ignored when SNA is selected for the **ConnectionType** property.

OverwriteHostFile Property

The **OverwriteHostFile** property on a **IEIGFileTransferCtl** object indicates whether a file transfer operation request to copy a file that will write over an existing file will be executed. This property affects file transfer operations using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a Long value representing an **eigAnswerYesNoEnum**. The default value for this property is **eigAnswerNo**.

Syntax

```
currentOverwriteFlag = object.OverwriteHostFile
object.OverwriteHostFile = eigAnswerYes
```

Remarks

The **OverwriteHostFile** property is used to set or return a flag that indicates whether a file transfer operation will overwrite an existing file. This property can be set to one of the following **eigAnswerYesNoEnum** constants:

Enumeration	Value	Description
eigAnswerYes	0	This value indicates that the file transfer operation will overwrite an existing host file if it exists.
eigAnswerNo	1	This value indicates that the file transfer operation will not overwrite an existing host file file if it exists.

The **OverwriteHostFile** property and the **AppendToEnd** property are mutually exclusive, so it is not possible to enable (set to yes) one of these properties before the opposing property is disabled (set to no). The **AppendToEnd** property takes precedence over the **OverwriteHostFile** property, since **AppendToEnd** defaults to yes and **OverwriteHostFile** defaults to no. Consequently, the order in which these properties are set will affect the outcome. For example, the following order will result in the properties being set correctly:

```
FileTransfer.AppendToEnd = eigAnswerNo         // correctly set to no
FileTransfer.OverwriteHostFile = eigAnswerYes  // correctly set to yes
```

In contrast, setting the properties in the improper order will cause the properties to be set incorrectly as follows:

```
FileTransfer.OverwriteHostFile = eigAnswerYes   // remains at no
// AppendToEnd defaults to eigAnswerYes, so this change is illegal
FileTransfer.AppendToEnd = eigAnswerNo          // correctly set to no
```

In this second case, the **OverwriteHostFile** property cannot be set to yes (enabled) until the **AppendToEnd** property is set to no (disabled).

Password Property

The **Password** property on an **IEIGFileTransferCtl** object indicates the password used for authentication. This property affects connection and authentication to a host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR and has no default value.

Syntax

```
currentUserPassword = object.Password
object.Password = Dialog.UserPassword
```

Remarks

The **Password** property is used to set or return the password used for authenticating the user on a host computer. A valid user name and password are normally required to access files on a host computer. The password is case-sensitive and is normally displayed as asterisks in a dialog box for security purposes.

PCCodePage Property

The **PCCodePage** property on an **IEIGFileTransferCtl** object indicates the code page to be used on the PC. This property affects how data conversion is handled using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a short value representing the PC code page for the data file. The default value for this property is 1252 representing a PC code page of Latin 1.

Syntax

```
currentPCCodePage = object.PCCodePage
object.PCCodePage = 1252
```

Remarks

The **PCCodePage** property is used to set or return the code page to be used on the PC. This value is used for data conversion of any character data in the host file to ANSI or Unicode character data in the local file on the Microsoft® Windows® computer.

PutFile Method

The **PutFile** method on an **IEIGFileTransferCtl** object copies a file from host storage to local storage using the Microsoft® Host File Transfer ActiveX® control.

Syntax

```
object.PutFile HostFile, LocalFile
```

Parameters

HostFile

This required parameter specifies the name of the host file that will be written to as a result of this operation. This parameter is a BSTR.

LocalFile

This required parameter specifies the path to a local file that will be copied to the host file. This parameter is a BSTR.

Remarks

The **PutFile** method can only be called after a connection has been established to the host (when the ConnectionState property is connected). The behavior of the **PutFile** method is affected by the values of the AppendToEnd, CreateIfNonExisting, and OverwriteHostFile properties.

Errors are returned for this method using the **ISupportErrorInfo** object.

RDBName Property

The **RDBName** property on an **IEIGFileTransferCtl** object indicates the remote database name and the host column description (HCD) file that describes the data types and data conversions used to transfer this file using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value and has no default value.

Syntax

```
currentRDbName = object.RDBName
object.RDBName = "Inventory"
```

Remarks

The **RDBName** property is used to set or return the name of the remote database name and the host column description (HCD) file that describes the data types and data conversions used to transfer this file. The HCD file describing the data should be located in the system subdirectory below the root directory where Host Integration Server was installed. Setup defaults to the following location: C:\Program Files\Microsoft Host Integration Server

When TCP/IP is selected for the **ConnectionType** property, the **RDBName** must also match the name of the remote database system.

RemoteLU Property

The **RemoteLU** property on an **IEIGFileTransferCtl** object indicates the remote LU alias for an APPC (SNA) connection type to the remote host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR string value representing the remote LU name. This property has no default value.

Syntax

```
currentRemoteLu = object.RemoteLU
object.RemoteLu = "Remote10"
```

Remarks

The **RemoteLU** property is used to set or return the remote LU alias. When LU 6.2 (SNA) is selected for the ConnectionType property, this property must match the name of the remote LU alias configured using SNA Manager.

This property is ignored when TCP/IP is selected for the **ConnectionType** property.

UserID Property

The **UserID** property on an **IEIGFileTransferCtl** object indicates the user name used for authentication. This property affects connection and authentication to a host computer using the Microsoft® Host File Transfer ActiveX® control. This property sets or returns a BSTR and has no default value.

Syntax

```
currentUserName = object.UserID
object.UserID = Dialog.UserName
```

Remarks

The **UserID** property is used to set or return the user name used for authenticating the user on a host computer. A valid user name and password are normally required to access files on a host computer. The user name is case-sensitive.

ADO Programmer's Reference

This section provides reference information about specific ActiveX Data Objects (ADO) methods, properties, and collections supported in Host Integration Server 2009 using the following:

- Microsoft OLE DB Provider for AS/400 and VSAM
- Microsoft OLE DB Provider for DB2
- Microsoft ODBC Driver for DB2

For general information about programming with the OLE DB Provider for AS/400 and VSAM, see the OLE DB Provider for AS/400 and VSAM Programmer's Guide in the OLE DB Providers Programmer's Guide section of the SDK.

For general information about programming with the OLE DB Provider for DB2, see the OLE DB Provider for DB2 Programmer's Guide.

For general information about programming with the ODBC Driver for DB2, see the ODBC Driver for DB2 Programmer's Guide in the OLE DB Providers Programmer's Guide section of the SDK.

In This Section

ActiveCommand Property (ADO)

The **ActiveCommand** property on a **Recordset** object indicates which **Command** object created the associated **Recordset** object. This property returns a variant that contains a **Command** object. The default is a **Null** object reference.

Syntax

```
currentCommand = recordset.ActiveCommand
```

Remarks

The **ActiveCommand** property is property is read-only. If a **Command** object was not used to create the current **Recordset**, then a **Null** object reference is returned.

Use this property to find the associated **Command** object when you are given only the resulting **Recordset** object.

ActiveConnection Property (ADO)

The **ActiveConnection** property on a **Command** or **Recordset** object indicates to which **Connection** object the specified **Command** or **Recordset** object currently belongs. This property sets or returns a String that contains the definition for a connection or a **Connection** object. The default is a **Null** object reference.

Syntax

```
command.ActiveConnection = connectionString
activeConnectionString = recordset.ActiveConnection
```

Remarks

The **ActiveConnection** property is used to determine the **Connection** object over which the specified **Command** object will execute or the specified **Recordset** will be opened.

For **Command** objects, the **ActiveConnection** property is read/write.

If you try to call the **Execute** method on a **Command** object before setting the **ActiveConnection** property to an open **Connection** object or valid connection string, an error occurs.

Under Visual Basic, setting the **ActiveConnection** property to **Nothing** disassociates the **Command** object from the current **Connection** and causes the OLE DB provider to release any associated resources on the data source. You can then associate the **Command** object with the same or another **Connection** object. Some providers allow you to change the **ActiveConnection** property setting from one **Connection** to another, without having to first set the property to **Nothing**.

Closing the **Connection** object with which a **Command** object is associated sets the **ActiveConnection** property to **Nothing**. Setting this property to a closed **Connection** object generates an error.

For open **Recordset** objects or for **Recordset** objects whose **Source** property is set to a valid **Command** object, the **ActiveConnection** property is read-only. Otherwise, it is read/write.

You can set the **ActiveConnection** property to a valid **Connection** object or to a valid connection string. In this case, the OLE DB provider creates a new **Connection** object using this definition and opens the connection. Additionally, the provider may set this property to the new **Connection** object to give you a way to access the **Connection** object for extended error information or to execute other commands.

If the *ActiveConnection* parameter of the **Open** method is used to open a **Recordset** object, the **ActiveConnection** property will inherit the value of the argument.

If the **Source** property of the **Recordset** object is set to a valid **Command** object variable, the **ActiveConnection** property of the **Recordset** inherits the setting of the **ActiveConnection** property of the **Command** object.

The information needed to establish a connection to a data source can be set in the **ActiveConnection** property of a **Recordset** object or passed as part of the **Open** method on a **Recordset** object in the *ActiveConnection* parameter. In either case, this information must be in a specific format for use with the Microsoft OLE DB Provider for AS/400 and VSAM, the Microsoft OLE DB Provider for DB2, or the Microsoft ODBC Driver for DB2. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons.

ActiveX Data Objects (ADO) supports several standard ADO-defined arguments for the **ActiveConnection** property as listed in the following table.

Argument	**Description**
Data Source	Specifies the name of the data source for the connection. This argument is optional when you use the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

File Name	Specifies the name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed. This argument is not supported by the OLE DB Provider for AS/400 and VSAM.
Location	The remote database name that is used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.
Password	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target host system and has appropriate access rights to the file.
Provider	Specifies the name of the provider to use for the connection. To use the OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB". To use the OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB". To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString because this value is the default for ADO.
User ID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target host system and has appropriate access rights to the file.

In This Section

ActiveConnection Property Support Using the OLE DB Provider for AS/400 and VSAM

ActiveConnection Property Support Using the OLE DB Provider for DB2

ActiveConnection Property Support Using the ODBC Driver for DB2

ActiveConnection Property Support Using the OLE DB Provider for AS/400 and VSAM

The Microsoft OLE DB Provider for AS/400 and VSAM also supports a number of provider-specific arguments, some of which have default values as specified in the table below. These arguments are listed in the following table.

Argument	Description
BinAsChar	This parameter indicates whether to process binary fields as character fields (default is 0; do not process binary fields as character fields).
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files.
HCD FileName	The fully qualified filename of the DDM host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the SNA system directory. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400.
Local LU	The name of the local LU alias configured in Host Integration Server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
RDB	The remote database name for OS/400. You only need to specify this value if it is different from the remote LU alias configured in Host Integration Server.
Repair Host Keys	This parameter indicates whether the OLE DB provider should repair any host key values set in the registry and defaults to false.
RemoteLU	The name of the remote logical unit (LU) alias configured in Host Integration Server.
StrictVal	This parameter indicates whether strict validation should be used and defaults to false.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** for use with the OLE DB Provider for AS/400 and VSAM follows:

```
Conn.Provider="SNAOLEDB"
Conn.ConnectionString = "User ID=USERNAME;Password=password",&_
  "LocalLU=LOCAL;RemoteLU=DATABASE",&_
  "ModeName=QPCSUPP;CCSID=37;PCCodePage=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic.

When opening a connection object in ActiveX® Data Objects (ADO) version 2.0, you must specify the Prompt connection property. For example, the following is valid with ADO version 1.5 and ADO version 2.0 and will prompt the user for **ConnectionString** properties:

```
Conn.ConnectionString = "Provider=SNAOLEDB
Conn.Properties("PROMPT")=adPromptAlways
Conn.Open
```

ActiveConnection Property Support Using the OLE DB Provider for DB2

The Microsoft OLE DB Provider for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by OLE DB Provider for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier OLE DB Provider for DB2 included with SNA Server 4.0.

The arguments supported by the OLE DB Provider for DB2 supplied with Host Integration Server are listed in the following table.

Argument	**Description**
BinAsChar	When this parameter is set to **true**, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This parameter defaults to **false**.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
DefSch	The name of the default schema (collection/owner) where the system catalogs resides. This parameter can be QSYS2;SYSIBM;SYSTEM; CURLIB; or USERID depending on platform. This parameter does not have a default value.
InitCat	This parameter is used as the first part of a three-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. In DB2/400, this parameter is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then one can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE. This parameter has no default value.
LocalLU	The name of the local logical unit (LU) alias configured in Host Integration Server.
ModeName	The Advanced Program-to-Program Communications (APPC) mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.

NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the computer. If this argument is omitted, the default value is set to Latin 1 (1252).
PkgCol	The name of the DRDA target collection (AS/400 library) where the OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2 uses packages to issue dynamic and static SQL statements. The OLE DB Provider will create packages dynamically in the location to which the user points using the Package Collection parameter.
RemoteLU	The name of the remote LU alias configured in Host Integration Server.
TPName	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPName is set to 0X07F9F9F9.
UOW	This parameter determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** using the OLE DB Provider for DB2 follows:

```
Conn.Provider="DB2OLEDB"
Conn.ConnectionString = "User ID=USERNAME;Password=password",&_
  "LocalLU=LOCAL;RemoteLU=DATABASE",&_
  "ModeName=QPCSUPP;CCSID=37;PcCodePage=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic.

ActiveConnection Property Support Using the ODBC Driver for DB2

The Microsoft ODBC Driver for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by ODBC Driver for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier ODBC Driver for DB2 included with SNA Server 4.0.

The arguments supported by the ODBC Driver for DB2 supplied with Host Integration Server are listed in the following table.

Argument	**Description**
BAC	When the *BinAsChar* parameter is set to **true** (1), the ODBC Driver for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The character code set identifier (CCSID) and PCCodePage values are required input parameters.
CCSID	The CCSID matching the DB2 data as represented on the remote computer. The CCSID property is required when processing binary data as character data. Unless the BinAsChar value is set, character data is converted based on the DB2 column CCSID and default ANSI code page. If this argument is omitted, this parameter defaults to U.S./Canada (37).
CP	The character code page to use on the computer. This parameter is required when processing binary data as character data. Unless the Binary as Character (BAC) value is set, character data is converted based on the default ANSI code page configured in Windows. If this argument is omitted, the default value is set to Latin 1 (1252).
DESC	A field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.
DS	The Default Schema parameter is the name of the Collection where the ODBC Driver for DB2 looks for catalog information. The Default Schema is the "SCHEMA" name for the target collection of tables and views. The ODBC driver uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (for example, ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, then the ODBC driver uses the USER_ID provided at login. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Obviously, this default is inappropriate in many cases, therefore it is essential that the Default Schema value in the data source be defined.
DSN	The data source name is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
LLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.

MN	When SNA is used for the Network Transport Library (NTL), the Mode Name field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to QPCSUPP.
NA	When TCP/IP is used for the Network Transport Library (NTL), the Network Address parameter indicates the IP address or the hostname alias of the host DB2 server.
NP	When TCP/IP is used for the Network Transport Library (NTL), the Network Port parameter indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default value is TCP/IP port 446.
NTL	The Network Transport Library parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA. If the default SNA is selected, then values for LLU, MN, and RLU are required. If TCP/IP is selected, then values for NetAddr and NetPort are required.
PC	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft ODBC Driver for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. The ODBC driver will create packages dynamically in the location to which the user points using the Package Collection parameter.
PDS	The Provider Data Source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
PROV	Specifies the name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString since this value is the default for ADO.
PWD	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. Note that this parameter is the same as the Parameter parameter.
RDB	The Remote Database Name parameter is used as the first part of a three-part, fully qualified DB2 table name. This parameter is referred to by different names depending on the DB2 platform. In DB2 on MVS and OS/390, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then a value can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE.
RLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.

TPN	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPN is set to 0X07F9F9F9.
UID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is the same as the User ID parameter.
UOW	Determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note
Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** using the ODBC Driver for DB2 supplied with Host Integration Server is as follows:

```
Conn.Provider="MSDASQL"
Conn.ConnectionString = "UID=USERNAME;PWD=password",&_
  "LLU=LOCAL;RLU=DATABASE",&_
  "MN=QPCSUPP;CCSID=37;CP=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note
The &_ character combination is used for continuing long lines in Visual Basic.

ActualSize Property (ADO)

The **ActualSize** property on a **Field** object indicates the actual length of a field's value. This property returns a Long value.

Syntax

```
size = field.ActualSize
```

Remarks

The **ActualSize** property is used to return the actual length of a **Field** object's value. For all fields, the **ActualSize** property is read-only. If ADO cannot determine the length of the **Field** object's value, the **ActualSize** property returns **adUnknown**.

The **ActualSize** and **DefinedSize** properties on a **Field** object can be different. For example, a **Field** object with a declared type of **adVarChar** (variable character data type) and a maximum length of 50 characters returns a **DefinedSize** property value of 50, but the **ActualSize** property value it returns is the length of the data stored in the field for the current record.

AddNew Method (ADO)

The **AddNew** method on a **Recordset** object creates a new record for an updatable **Recordset** object.

Syntax

```
recordset.AddNewFields, Values
```

Parameters

Fields

This optional parameter specifies a single name or an array of names or ordinal positions of the fields in the new record.

Values

This optional parameter specifies a single value or an array of values for the fields in the new record. If *Fields* is an array, *Values* must also be an array with the same number of members; otherwise, an error occurs. The order of field names must match the order of field values in each array.

Remarks

The **AddNew** method is used to create and initialize a new record. The **Supports** method can be used with **adAddNew** to verify whether records can be added to the current **Recordset** object.

After the **AddNew** method is called, the new record becomes the current record and remains current after the **Update** method is called. If the **Recordset** object does not support bookmarks, you may not be able to access the new record after you move to another record. Depending on your cursor type, you may need to call the **Requery** method to make the new record accessible.

If **AddNew** is called while editing the current record or while adding a new record, ADO calls the **Update** method to save any changes and then creates the new record.

The behavior of the **AddNew** method depends on the updating mode of the **Recordset** object and whether or not the *Fields* and *Values* arguments are passed.

In immediate update mode, the OLE DB Provider writes changes to the underlying data source after the **Update** method is called. In immediate update mode, calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The OLE DB Provider caches any field value changes locally. Calling the **Update** method posts the new record to the database and resets the **EditMode** property to **adEditNone**. If the *Fields* and *Values* arguments are passed, ADO immediately posts the new record to the database (no **Update** call is necessary) and the **EditMode** property value does not change (**adEditNone**).

In batch update mode, the OLE DB Provider caches multiple changes and writes them to the underlying data source only when the **UpdateBatch** method is called. In batch update mode, calling the **AddNew** method without arguments sets the **EditMode** property to **adEditAdd**. The OLE DB Provider caches any field value changes locally. Calling the **Update** method adds the new record to the current **Recordset** object and resets the **EditMode** property to **adEditNone**, but the OLE DB Provider does not post the changes to the underlying database until the **UpdateBatch** method is called. If the *Fields* and *Values* arguments are passed, ADO sends the new record to the provider for storage in a cache and the **UpdateBatch** method must be called to post the new record to the underlying database.

AppendChunk Method (ADO)

The **AppendChunk** method on a **Field** object appends data to a large text or binary data **Field** object.

Syntax

```
field.AppendChunkData
```

Parameters

Data

This parameter specifies a Variant containing the data to be appended to the **Field** object.

Remarks

The **AppendChunk** method is used on a **Field** object to fill it with long binary or character data. In situations where system memory is limited, the **AppendChunk** method can be used to manipulate long values in portions rather than in their entirety.

If the **adFldLong** bit in the **Attributes** property of a **Field** object is set to **True**, the **AppendChunk** method can be used for that field.

The first **AppendChunk** call on a **Field** object writes data to the field, overwriting any existing data. Subsequent **AppendChunk** calls add to existing data. If you are appending data to one field and then set or read the value of another field in the current record, ActiveX® Data Objects (ADO) assumes that you are finished appending data to the first field. If the **AppendChunk** method is called on the first field again, ADO interprets the call as a new **AppendChunk** operation and overwrites the existing data. Accessing fields in other **Recordset** objects (that are not clones of the first **Recordset** object) will not disrupt **AppendChunk** operations.

If there is no current record when the **AppendChunk** method is called on a **Field** object, an error occurs.

Attributes Property (ADO)

The **Attributes** property on a **Field** object or a **Property** object in a **Properties** collection indicates one or more characteristics of an object. This property returns a Long value.

Syntax

```
attribute = field.Attributes
```

Remarks

The **Attributes** property is used to return characteristics of **Field** objects or **Property** objects.

For a **Field** object, the **Attributes** property is read-only and its value can be the sum of any one or more of the **FieldAttributeEnum** values. The allowable **FieldAttributeEnum** values can be one of the constants in the following table.

Enumeration	Value	Description
adFldMayDefer	0x2	This value indicates that the field is deferred, that is, the field values are not retrieved from the data source with the whole record, but only when you explicitly access them.
adFldUpdatable	0x4	This value indicates that you can write to the field.
adFldUnknownUpdatable	0x8	This value indicates that the provider cannot determine if you can write to the field.
adFldFixed	0x10	This value indicates that the field contains fixed-length data.
adFldIsNullable	0x20	This value indicates that the field accepts Null values.
adFldMayBeNull	0x40	This value indicates that you can read Null values from the field.
adFldLong	0x80	This value indicates that the field is a long binary field. This value also indicates that the **AppendChunk** and **GetChunk** methods on the **Field** object can be used.
adFldRowID	0x100	This value indicates that the field contains some kind of record identifier (record number, unique identifier, and so on).
adFldRowVersion	0x200	This value indicates that the field contains some kind of time or date stamp (often used to track updates).
adFldCacheDeferred	0x1000	This value indicates that the provider caches field values and that subsequent reads are done from the cache.

For a **Property** object, the **Attributes** property is read-only and its value can be the sum of any one or more of the **PropertyAttributesEnum** values. The allowable **PropertyAttributesEnum** values can be one of the constants in the following table.

Enumeration	Value	Description
adPropNotSupported	0	This value indicates that the property is not supported by the provider.

adPropRequired	0x1	This value indicates that the user must specify a value for this property before the data source is initialized.
adPropOptional	0x2	This value indicates that the user does not need to specify a value for this property before the data source is initialized.
adPropRead	0x200	This value indicates that the user can read the property.
adPropWrite	0x400	This value indicates that the user can set the property.

BOF Property (ADO)

The **BOF** property on a **Recordset** object indicates that the current record position is before the first record in a **Recordset** object. This property returns a Boolean value.

Syntax

```
IsBOF = recordset.BOF
```

Remarks

The **BOF** property is used to determine whether a **Recordset** object contains records or whether you have gone beyond the limits of a **Recordset** object when you move from record to record.

The **BOF** property returns **True** if the current record position is before the first record and **False** if the current record position is on or after the first record.

If the **BOF** property is **True**, there is no current record.

If a **Recordset** object is opened containing no records, both the **BOF** and **EOF** properties are set to **True** and the **Recordset** object's **RecordCount** property setting is zero. When a **Recordset** object is opened that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

If the last remaining record in the **Recordset** object is deleted, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

This table below indicates which **Move** methods are allowed with different combinations of the **BOF** and **EOF** properties.

	MoveFirst MoveLast	**MovePrevious Move < 0**	**Move 0**	**MoveNext Move > 0**
BOF=True EOF=False	Allowed	Error	Error	Allowed
BOF=False EOF=True	Allowed	Allowed	Error	Error
Both True	Error	Error	Error	Error
Both False	Allowed	Allowed	Allowed	Allowed

Note

Executing a **Move** 0 method when the **BOF** property is **True** does not currently generate an error using the OLE DB Provider for AS/400 and VSAM.

Note

Allowing a **Move** method does not guarantee that the method will successfully locate a record; it only means that calling the specified **Move** method will not generate an error.

The following table shows what happens to the **BOF** and **EOF** property settings when various **Move** methods are called but are unable to successfully locate a record.

	BOF property	**EOF property**
MoveFirst MoveLast	Set to **True**	Set to **True**
Move 0	No change	No change
MovePrevious Move < 0	Set to **True**	No change
MoveNext Move > 0	No change	Set to **True**

Bookmark Property (ADO)

The **Bookmark** property on a **Recordset** object returns a bookmark that uniquely identifies the current record in a **Recordset** object or sets the current record in a **Recordset** object to the record identified by a valid bookmark. This property sets or returns a **Variant** expression that evaluates to a valid bookmark.

Syntax

```
FirstBookmark = recordset.Bookmark
recordset.Bookmark = PreviousBookmark
```

Remarks

The **Bookmark** property is used to save the position of the current record and return to that record at any time. Bookmarks are available only in **Recordset** objects (host tables) that support the bookmark feature.

When a **Recordset** object is opened, each of its records has a unique bookmark. To save the bookmark for the current record, assign the value of the **Bookmark** property to a variable. To quickly return to that record at any time after moving to a different record, set the **Recordset** object's **Bookmark** property to the value of that variable.

The user may not be able to view the value of the bookmark. Also, users should not expect bookmarks to be directly comparable—two bookmarks that refer to the same record may have different values.

If the **Clone** method is used to create a copy of a **Recordset** object, the **Bookmark** property settings for the original and the duplicate **Recordset** objects are identical and you can use them interchangeably. However, you cannot use bookmarks from different **Recordset** objects interchangeably, even if they were created from the same source or command.

Using the OLE DB Provider for AS/400 and VSAM, only some data sources can be bookmarked. Calling the **Supports** method with the **adBookmark** argument will indicate if the data source (table) can be bookmarked.

CacheSize Property (ADO)

The **CacheSize** property on a **Recordset** object indicates the number of records from a **Recordset** object that are cached locally in memory. This property sets or returns a Long value that must be greater than zero. The default value for the **CacheSize** property is 1.

Syntax

```
previousSize = recordset.CacheSize
recordset.CacheSize = 1
```

Remarks

The **CacheSize** property is used to control how many records the provider keeps in its buffer and how many to retrieve at one time into local memory. For example, if the **CacheSize** is 10, after first opening the **Recordset** object, the provider retrieves the first 10 records into local memory. As you move through the **Recordset** object, the provider returns the data from the local memory buffer. As soon as you move past the last record in the cache, the provider retrieves the next 10 records from the data source into the cache.

The value of the **CacheSize** property can be adjusted during the life of the **Recordset** object, but changing this value only affects the number of records in the cache after subsequent retrievals from the data source. Changing the property value alone will not change the current contents of the cache.

If there are fewer records to retrieve than the **CacheSize** property specifies, the provider returns the remaining records; no error occurs.

A **CacheSize** setting of zero is not allowed and returns an error. Non-bookmarkable files cannot have the **CacheSize** property set to greater than one, or an error will occur.

It is strongly recommended that a **CacheSize** of 1 be used with the OLE DB Provider for AS/400 and VSAM. If the **CacheSize** is set greater than 1, it is possible for the local data cached in memory to be out of date from changes made by other users on the host.

CancelBatch Method (ADO)

The **CancelBatch** method on a **Recordset** object cancels a pending batch update.

Syntax

```
recordset.CancelBatchAffectedRecords
```

Parameters

AffectedRecords

This optional parameter specifies an **AffectEnum** value that determines how many records the **CancelBatch** method will affect. The **AffectEnum** value can be one of the constants in the following table.

Enumeration	Value	Description
adAffectCurrent	1	This value cancels pending updates only for the current record.
adAffectGroup	2	This value cancels pending updates for records that satisfy the current Filter property setting. You must set the **Filter** property to one of the valid predefined constants to use this option.
adAffectAll	3	This value cancels pending updates for all the records in the **Recordset** object, including any hidden by the current **Filter** property setting. This value is the default.

Remarks

The **CancelBatch** method is used to cancel any pending updates in a **Recordset** object in batch update mode. If the **Recordset** object is in immediate update mode, calling **CancelBatch** without **adAffectCurrent** generates an error.

If you are editing the current record or are adding a new record when **CancelBatch** is called, ActiveX® Data Objects (ADO) first calls the CancelUpdate Method to cancel any cached changes, and then all pending changes in the recordset are canceled.

It is possible that the current record will be indeterminable after a **CancelBatch** call, especially if you were in the process of adding a new record. For this reason, it is prudent to set the current record position to a known location in the recordset after the **CancelBatch** method is called. For example, call the **MoveFirst** method.

If the attempt to cancel the pending updates fails because of a conflict with the underlying data (for example, a record has been deleted by another user), the provider returns warnings to the **Errors** collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. The Filter Property (**adFilterAffectedRecords**) and the **Status** property can be used to locate records with conflicts.

CancelUpdate Method (ADO)

The **CancelUpdate** method on a **Recordset** object cancels any changes made to the current record or to a new record prior to calling the **Update** method.

Syntax

```
recordset.CancelUpdate
```

Parameters

None.

Remarks

The **CancelUpdate** method is used to cancel any changes made to the current record or to discard a newly added record. You cannot undo changes to the current record or to a new record after the **Update** method is called unless the changes are part of a batch update that you can cancel with the CancelBatch Method.

If you are adding a new record when the **CancelUpdate** method is called, the record that was current prior to the AddNew Method call becomes the current record again.

If you have not changed the current record or added a new record, calling the **CancelUpdate** method generates an error.

Clear Method (ADO)

The **Clear** method on a **Collection** object removes all of the objects in a collection.

Syntax

```
collection.Clear
```

Parameters

None.

Remarks

The **Clear** method is used on the **Errors** collection to remove all existing **Error** objects from the collection. When an error occurs, ActiveX® Data Objects (ADO) automatically clears the **Errors** collection and fills it with **Error** objects based on the new error. However, some properties and methods return warnings that appear as **Error** objects in the **Errors** collection but do not halt a program's execution. Before calling the **Resync**, **UpdateBatch**, or CancelBatch Methods on a **Recordset** object or before setting the Filter Property on a **Recordset** object, call the **Clear** method on the **Errors** collection. Doing so enables you to read the **Count** property of the **Errors** collection to test for returned warnings as a result of these specific calls.

Clone Method (ADO)

The **Clone** method on a **Recordset** object creates a duplicate **Recordset** object from an existing **Recordset** object.

Syntax

```
rstDuplicate = rstOriginal.Clone
```

Parameters

rstDuplicate

This object variable specifies the duplicate **Recordset** object to be created.

rstOriginal

This object variable specifies the **Recordset** object to be duplicated.

Remarks

The **Clone** method is used on a **Recordset** object to create multiple, duplicate **Recordset** objects, particularly if you want to be able to maintain more than one current record in a given set of records. Using the **Clone** method is more efficient than creating and opening a new **Recordset** object with the same definition as the original.

The current record of a newly created clone is set to the first record.

Changes made to one **Recordset** object are visible in all of its clones regardless of cursor type. However, after you execute **Requery** on the original **Recordset**, the clones will no longer be synchronized to the original.

Closing the original **Recordset** does not close its copies; closing a copy does not close the original or any of the other copies.

You can only clone a **Recordset** object that supports bookmarks. Bookmark values are interchangeable; that is, a bookmark reference from one **Recordset** object refers to the same record in any of its clones.

Close Method (ADO)

The **Close** method on a **Connection** or **Recordset** object closes an open object and any dependent objects.

Syntax

```
recordSet.Close
```

Parameters

None

Remarks

The **Close** method is used to close either a **Connection** object or a **Recordset** object to free any associated system resources. Closing an object does not remove it from memory; you may change its property settings and open it again later. To completely eliminate an object from memory, set the object variable to **Nothing**.

Using the **Close** method to close a **Connection** object also closes any active **Recordset** objects associated with the connection. A **Command** object associated with the **Connection** object you are closing will persist, but it will no longer be associated with a **Connection** object, that is, its **ActiveConnection** property will be set to **Nothing**.

You can later call the **Open** method to reestablish the connection to the same or another data source. While the **Connection** object is closed, calling any methods that require an open connection to the data source generates an error. Closing a **Connection** object while there are open **Recordset** objects on the connection rolls back any pending changes in all of the **Recordset** objects.

Using the **Close** method to close a **Recordset** object releases the associated data and any exclusive access you may have had to the data through this particular **Recordset** object. You can later call the **Open** method to reopen the recordset with the same or modified attributes. While the **Recordset** object is closed, calling any methods that require a live cursor generates an error.

If an edit is in progress while in immediate update mode, calling the **Close** method generates an error. The **Update** or CancelUpdate Methods should be called first. If you close the **Recordset** object during batch updating, everything changes since the last **UpdateBatch** call is lost.

CommandText Property (ADO)

The **CommandText** property on a **Command** object contains the text of a command that you want to issue against a provider. This property sets or returns a String value containing a provider command, such as an AS/400 Command Language (CL) command for execution by the remote OS/400 DDM target server or an SQL command for execution on a DB2 database server. The default value for the **CommandText** property is a zero-length string.

Syntax

```
previousCommandtext = command.CommandText
command.CommandText= "EXEC COMMAND DDMCmd"
```

Remarks

The **CommandText** property is used to set or return the text of a **Command** object. When used with the OLE DB Provider for AS/400 and VSAM, the text can be an AS/400 CL command for execution by the remote OS/400 DDM target server or a request to open a table on a host (a remote DDM Server). When used with the OLE DB Provider for DB2, the text can be an SQL command for execution or a call to a stored procedure.

If the **Prepared** property of the **Command** object is set to **True** and the **Command** object is bound to an open connection when you set the **CommandText** property, ActiveX® Data Objects (ADO) prepares the query when you call the **Execute** or **Open** methods.

Depending on the **CommandType** property setting, ADO may alter the **CommandText** property. The **CommandText** property can be read at any time to see the actual command text that ADO will use during execution.

The **CommandText** property defines the text version of a command. The syntax for the string in the **CommandText** property when used with the OLE DB Provider for AS/400 and VSAM is as follows:

```
EXEC COMMAND DDMCmd
```

where *DDMCmd* represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands allow you to request functions from the OS/400 operating system. Some examples are the DLTF (Delete File) or DSPFFD (Display File Description) commands. These are the same commands that could be issued on the command line if you were connected to an AS/400 via a 5250 terminal session. See the 'OS/400 CL Reference for your platform for a detailed list of possible commands.

With the OLE DB Provider for AS/400 and VSAM, the **Command** object can also be used to open a data file after a **Connection** object has been opened and the **ActiveConnection** property has been set to this open connection. The **CommandText** property defines the data file to open. When used with the OLE DB Provider for AS/400 and VSAM, the syntax for the **CommandText** property string in this case is as follows:

```
EXEC OPEN DataSetName
```

where *DataSetName* represents a valid data file or library member on the host. If you open a host data file from a **Command** object, then the data file is opened as read-only. This results from the limitation that no argument or option is passed by ADO that supplies a parameter describing whether the data set should be opened as read-only or updatable.

The syntax for the string in the **CommandText** property when used with the OLE DB Provider for DB2 can be one of the following:

```
EXEC SQLStatement
```

where SQLStatement represents a valid SQL statement supported by DB2.

```
CALL StoredProcedure
```

where StoredProcedure represents a valid DB2 stored procedure on the database server.

The **CommandType** property specifies the type of command described in the **CommandText** property prior to execution in order to optimize performance. The **CommandType** property must be set to **adCmdText** for use with the OLE DB Provider

for AS/400 and VSAM or the OLE DB Provider for DB2.

CommandType Property (ADO)

The **CommandType** property on a **Command** object Indicates the type of a **Command** object. This property sets or returns a **CommandTypeEnum** value.

Syntax

```
oldType = command.CommandType
command.CommandType= newType
```

Remarks

The **CommandType** property is used to set or return the type of a **Command** object. This property specifies a **CommandTypeEnum** value that can be one of the constants in the following table.

Enumeration	Value	Description
adCmdUnspecified	-1	This value indicates that the **CommandType** property has been unspecified.
adCmdText	1	This value evaluates the **CommandText** property as a textual definition of a command or stored procedure call.
adCmdTable	2	This value evaluates the **CommandText** property as a table name. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adCmdStoredProc	4	This value evaluates the **CommandText** property as a stored procedure. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2. See remarks below regarding using stored procedures using the OLE DB Provider for DB2.
adCmdUnknown	8	This value indicates that the type of command in a **CommandText** property is not known. This is the default value.

The OLE DB Provider for AS/400 and VSAM and the OLE DB Provider for DB2 only support the **adCmdText** type for the **CommandType** property. If any other value for the **CommandType** property is set, errors will occur.

The OLE DB Provider for DB2 supports calling DB2 stored procedures. An application must use the CALL keyword before the SQL statement in order to execute a stored procedure. When using ADO, a **CommandType** property of **adCmdStoredProc** cannot be used for executing a stored procedure since ActiveX® Data Objects (ADO) inserts an EXEC not CALL keyword before the command text. In order to execute a stored procedure using ADO, the **CommandType** property should be set to **adCmdText** and the CALL keyword should be used before the SQL statement containing the stored procedure to be executed.

ConnectionString Property (ADO)

The **ConnectionString** property on a **Connection** object contains the information used to establish a connection to a data source. This property sets or returns a string value.

Syntax

```
oldString = connection.ConnectionString
connection.ConnectionString = newString
```

Remarks

The **ConnectionString** property is used to specify a data source by passing a detailed connection string containing a series of *argument = value* statements separated by semicolons.

Microsoft ActiveX Data Objects (ADO) supports several standard arguments for the **ConnectionString** property. Any other arguments are passed directly to the provider without any processing by ADO. This information must be in a specific format for use with the Microsoft OLE DB Provider for AS/400 and VSAM, the Microsoft OLE DB Provider for DB2 or the Microsoft ODBC Driver for DB2. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons. ADO supports several standard ADO-defined arguments for the **ConnectionString** property as listed in the following table.

Argument	**Description**
Data Source	This argument specifies the name of the data source for the connection. This argument is optional when using the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
File Name	This argument specifies the name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed. This argument is not supported by the OLE DB Provider for AS/400 and VSAM.
Location	The remote database name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.
Password	This argument specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target host system and has appropriate access rights to the file.
Provider	This argument specifies the name of the provider to use for the connection. To use the OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB". To use the OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB". To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString since this value is the default for ADO.
Remote Provider	This argument specifies the name of a provider to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
Remote Server	This argument specifies the path name of a server to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

User ID	This argument specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target host system and has appropriate access rights to the file.

Note
Not all of these parameters are required. The user can also be prompted for this information.

The OLE DB Provider for AS/400 and VSAM also support a number of provider-specific arguments, some of which default have default values as specified in the following table.

Argument	**Description**
BinAsChar	This parameter indicates whether to process binary fields as character fields (default is 0; do not process binary fields as character fields).
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files.
HCDFileName	The fully qualified filename of the DDM host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the SNA system directory. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400.
Local LU	The name of the local LU alias configured in Host Integration Server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
RDB	The remote database name for OS/400. You only need to specify this value if it is different from the remote LU alias configured in Host Integration Server.
Repair Host Keys	This parameter indicates whether the OLE DB provider should repair any host key values set in the registry and defaults to false.

RemoteLU	The name of the remote LU alias configured in Host Integration Server.
StrictVal	This parameter indicates whether strict validation should be used and defaults to false.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** for use with the OLE DB Provider for AS/400 and VSAM follows:

```
Conn.Provider="SNAOLEDB"
Conn.ConnectionString = "User ID=USERNAME;Password=password",&_
  "LocalLU=LOCAL;RemoteLU=DATABASE",&_
  "ModeName=QPCSUPP;CCSID=37;PCCodePage=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic.

When opening a connection object in ADO 2.0, you must specify the Prompt connection property. For example, the following is valid with ADO 1.5 and ADO 2.0 and will prompt the user for **ConnectionString** properties:

```
Conn.ConnectionString = "Provider=SNAOLEDB
Conn.Properties("PROMPT")=adPromptAlways
Conn.Open
```

The OLE DB Provider for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by OLE DB Provider for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier OLE DB Provider for DB2 included with SNA Server 4.0.

The arguments supported by the OLE DB Provider for DB2 supplied with Host Integration Server 2009 are listed in the following table.

Argument	**Description**
BinAsChar	When this parameter is set to true, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This parameter defaults to false.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).

DefSch	The name of the default schema (collection/owner) where the system catalogs resides. This parameter can be QSYS2, SYSIBM, SYSTEM, CURLIB, or USERID depending on platform. This parameter does not have a default value.
InitCat	This parameter is used as the first part of a 3-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. In DB2/400, this parameter is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then one can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE. This parameter has no default value.
LocalLU	The name of the local LU alias configured in Host Integration Server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
PkgCol	The name of the DRDA target collection (AS/400 library) where the Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2 uses packages to issue dynamic and static SQL statements. The OLE DB Provider will create packages dynamically in the location to which the user points using the Package Collection parameter.

RemoteLU	The name of the remote LU alias configured in Host Integration Server.
TPName	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPName is set to 0X07F9F9F9.
UOW	This parameter determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

The ODBC Driver for DB2 supports a number of provider-specific arguments, some of which have default values as specified in the following tables. The arguments supported by ODBC Driver for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier ODBC Driver for DB2 included with SNA Server 4.0.

The arguments supported by the ODBC Driver for DB2 supplied with Host Integration Server 2009 are listed in the following table.

Argument	**Description**
BAC	When the *BinAsChar* parameter is set to **true** (1), the ODBC Driver for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The CCSID and PCCodePage values are required input parameters.
CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote computer. The CCSID property is required when processing binary data as character data. Unless the BinAsChar value is set, character data is converted based on the DB2 column CCSID and default ANSI code page. If this argument is omitted, this parameter defaults to U.S./Canada (37).
CP	The character code page to use on the PC. This parameter is required when processing binary data as character data. Unless the Binary as Character (BAC) value is set, character data is converted based on the default ANSI code page configured in Windows. If this argument is omitted, the default value is set to Latin 1 (1252).

DESC	A field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.
DS	The Default Schema parameter is the name of the Collection where the ODBC Driver for DB2 looks for catalog information. The Default Schema is the "SCHEMA" name for the target collection of tables and views. The ODBC driver uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (e.g., ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, then the ODBC driver uses the USER_ID provided at login. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Obviously, this default is inappropriate in many cases, therefore it is essential that the Default Schema value in the data source be defined.
DSN	The data source name is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
LLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
MN	When SNA is used for the Network Transport Library (NTL), the Mode Name field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to QPCSUPP.
NA	When TCP/IP is used for the Network Transport Library (NTL), the Network Address parameter indicates the IP address or the hostname alias of the host DB2 server.
NP	When TCP/IP is used for the Network Transport Library (NTL), the Network Port parameter indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default value is TCP/IP port 446.
NTL	The Network Transport Library parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA. If the default SNA is selected, then values for LLU, MN, and RLU are required. If TCP/IP is selected, then values for NetAddr and NetPort are required.
PC	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft ODBC Driver for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. The ODBC driver will create packages dynamically in the location to which the user points using the Package Collection parameter.

PDS	The Provider Data Source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
PROV	Specifies the name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString since this value is the default for ADO.
PWD	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. Note that this parameter is the same as the Parameter parameter.
RDB	The remote database name parameter is used as the first part of a three-part, fully qualified DB2 table name. This parameter is referred to by different names depending on the DB2 platform. In DB2 on MVS and OS/390, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then a value can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE.
RLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
TPN	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPN is set to 0X07F9F9F9.
UID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is the same as the User ID parameter.
UOW	Determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

A sample **ConnectionString** using the ODBC Driver for DB2 supplied with Host Integration Server is as follows:

```
Conn.Provider="MSDASQL"
Conn.ConnectionString = "UID=USERNAME;PWD=password",&_
  "LLU=LOCAL;RLU=DATABASE",&_
```

```
  "MN=QPCSUPP;CCSID=37;CP=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

Note

The &_ character combination is used for continuing long lines in Visual Basic.

After the **ConnectionString** property is set and the **Connection** object is opened, the provider may alter the contents of the property, for example, by mapping the ADO-defined argument names to their provider equivalents. Using the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2, three items are stripped from the **ConnectionString** after a **Connection** object is opened: Data Source, User ID, and Password.

The **ConnectionString** property automatically inherits the value used for the *ConnectionString* argument of the **Open** method on a **Connection** object, so you can override the current **ConnectionString** property during the **Open** method call. Therefore, the **ConnectionString** property of the **Connection** object can be set before opening the **Connection** object, or the *ConnectionString* parameter can be used to set or override the current connection parameters during the **Open** method call.

The **ConnectionString** property is read/write when the connection is closed and read-only when it is open.

If user and password information is set in both the **ConnectionString** property and in the optional *UserID* and *Password* parameters to the **Open** method, the results may be unpredictable. Such information should only be passed in either the **ConnectionString** property (or the *ConnectionString* parameter to the **Open** method call) or in the *UserID* and *Password* parameters.

There are a number of different ways to open a connection. The **Open** method can pass all of the appropriate connection information as part of the *ConnectionString* parameter or by setting the **ConnectionString** property of the **Connection** object, if this information is known in advance. The syntax in this case using the **ConnectionString**property is as follows:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.Provider="SNAOLEDB"
connection.ConnectionString = "User ID=USERNAME;Password=password;Local LU=LOCAL;Remote LU=
DATABASE;ModeName=QPCSUPP;CCSID=37;CodePage=437"
Conn.Properties("PROMPT")=adPromptNever
Conn.Open
```

There is a lack of spaces after the semicolons in the string. If spaces are inserted after the semicolons, an error will occur.

The simplest form of a **ConnectionString** property that contains all necessary information is as follows:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.ConnectionString = "Provider=SNAOLEDB;Data Source=REMLU;User ID=USERNAME;Passwor
d=password;ModeName=QPCSUPP"
```

Note

The User ID and Password must be included. Note the lack of spaces after the semicolons in the string. If spaces are inserted after the semicolons, an error will occur.

In the case where you would like the user to input the connection information, the following syntax can be used. This syntax does not specify any connection information except the provider, which is always required unless this is set in the **ConnectionString** or **Provider** property of the **Connection** object:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.ConnectionString = "Provider=SNAOLEDB"
Conn.Properties("PROMPT")=adPromptAlways
connection.Open
```

This method of invoking the **Open** method automatically causes a dialog box to appear asking the user for the user name, password, and other necessary information.

CursorLocation Property (ADO)

The **CursorLocation** property on a **Connection** object or **Recordset** object indicates the location of the cursor engine. This property sets or returns a Long value representing a **CursorLocationEnum**.

Syntax

```
cursor = connection.CursorLocation
connection.CursorLocation= adUseServer
```

Remarks

The **CursorLocation** property is used to set or return the location of the cursor. This property can be set to one of the **CursorLocationEnum** constants listed in the following table.

Enumeration	Value	Description
adUseNone	1	This value indicates no cursor location. This value is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM.
adUseServer	2	This value indicates that the data provider or driver-supplied cursor is used.
adUseClient	3	This value indicates that a client-side cursor supplied by a local cursor library is to be used.
adUseClientBatch	3	For backward compatibility, this value indicates that a client-side cursor supplied by a local cursor library is to be used.

This property setting only affects connections established after the property has been set. Changing the **CursorLocation** property has no effect on existing connections.

This property is read/write on a **Connection** object or a closed **Recordset**, object and read-only on an open **Recordset**.

If the **CursorLocation** property is set to **adUseClient**, the recordset will be accessible as read-only, and recordset updates to the host are not possible. When the CursorLocation property is set to **adUseClient** (use the client cursor engine), the **Find** method, **Filter** property, and **Sort** property will work if MDAC 2.0 or higher is installed, but will not work properly with earlier versions of ADO.

CursorType Property (ADO)

The **CursorType** property on a **Recordset** object indicates the type of the cursor engine. This property sets or returns a Long value representing a **CursorTypeEnum**.

Syntax

```
oldType = recordset.CursorType
recordset.CursorType= newType
```

Remarks

The **CursorType** property is used to set or return the type of the cursor that the provider should use when opening the **Recordset**. This property can be one of the enumerated values for **CursorTypeEnum** listed in the following table.

Enumeration	Value	Description
adOpenUnspecified	-1	This indicates an unspecified value for the *CursorType*. This value is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adOpenForwardOnly	0	Specifying this value opens a forward-only-type cursor. This *CursorType* is identical to a static cursor, except that you can only scroll forward through records. This improves performance when only one pass through a **Recordset** is needed. This value is not supported by the Microsoft OLE DB Provider for AS/400 and VSAM.
adOpenKeyset	1	Specifying this value opens a keyset-type cursor. This *CursorType is* similar to a dynamic cursor with a few exceptions. Records that other users delete are inaccessible from your **Recordset**. Data changes to existing records by other users are still visible, but records added by other users are not visible (cannot be seen). This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adOpenDynamic	2	Specifying this value opens a dynamic-type cursor. Additions, changes, and deletions by other users are visible, and all types of movement through the recordset are allowed, except for bookmarks if the provider does not support them. A dynamic cursor is the only *CursorType* supported by the OLE DB Provider for AS/400 and VSAM.
adOpenStatic	3	Specifying this value opens a static-type cursor. A static cursor provides a static copy of a set of records that can be used to find data or generate reports. Additions, changes, or deletions by other users are not visible with a static cursor. This value is not supported by the OLE DB Provider for AS/400 and VSAM.

Note that the **Open** method on a **Recordset** object defaults this property to **adOpenForwardOnly**,a value that is mapped to **adOpenDynamic** by the OLE DB provider for AS/400 and VSAM.

This property setting only affects connections established after the property has been set. Changing the **CursorType** property has no effect on existing connections. The **CursorType** property is read/write when the **Recordset** is closed and read-only when it is open.

If a provider does not support the requested cursor type, the provider may return another cursor type. The **CursorType** property will change to match the actual cursor type in use when the **Recordset** object is open. To verify whether a specific returned cursor is supported, use the **Supports** method. When using the OLE DB Provider for AS/400 and VSAM, the **Supports** method returns **adMovePrevious** as true with **CursorType** set to **adOpenDynamic**. After you close the **Recordset**, the **CursorType** property reverts to its original setting.

DefinedSize Property (ADO)

The **DefinedSize** property on a **Field** object indicates the defined size of a field object. This property returns a Long value that reflects the defined size of a field as a number of bytes.

Syntax

```
size = field.DefinedSize
```

Remarks

The **DefinedSize** property is used to return the data capacity or length of a **Field** object. For all fields, the **DefinedSize** property is read-only. If ActiveX® Data Objects (ADO) cannot determine the length of the **Field** object, the **ActualSize** property returns **adUnknown**.

The **ActualSize** and **DefinedSize** properties on a field object can be different. For example, a **Field** object with a declared type of **adVarChar** (variable character data type) and a maximum length of 50 characters returns a **DefinedSize** property value of 50, but the **ActualSize** property value it returns is the length of the data stored in the field for the current record.

Delete Method (ADO)

The **Delete** method on a **Recordset** object deletes the current record or a group of records.

Syntax

```
recordSet.Delete AffectedRecords
```

Parameters

AffectedRecords

This optional parameter specifies an **AffectEnum** value that determines how many records the **Delete** method will affect. The **AffectEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adAffectCurrent	1	This value deletes only the current record.
adAffectGroup	2	This value deletes the records that satisfy the current Filter Property setting. You must set the **Filter** property to one of the valid predefined constants to use this option.

Remarks

The **Delete** method is used to mark the current record or a group of records in a **Recordset** object for deletion. If the **Recordset** object does not allow record deletion, an error occurs. If you are in immediate update mode, deletions occur in the database immediately. Otherwise, the records are marked for deletion from the cache and the actual deletion happens when the **UpdateBatch** method is called. (Use the **Filter** property to view the deleted records.)

Retrieving field values from the deleted record generates an error. After deleting the current record, the deleted record remains current until you move to a different record. After you move away from the deleted record, it is no longer accessible.

If you are in batch update mode, use the CancelBatch Method to cancel a pending deletion or group of pending deletions.

If the attempt to delete records fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the data provider returns warnings to the **Errors** collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records.

Description Property (ADO)

The **Description** property on an **Error** object describes the **Error** object. This property returns a String value that contains a description of the error.

Syntax

```
errorString = currentError.Description
```

Remarks

The **Description** property on an **Error** object is used to obtain a short description of the error. Applications can display this property to alert the user to an error that the application does not want to handle. The string will come from either ActiveX® Data Objects (ADO) or a data provider such as the Microsoft® OLE DB Provider for AS/400 and VSAM, the Microsoft OLE DB Provider for DB2, or the Microsoft ODBC Driver for DB2. The provider is responsible for passing specific error text to ADO.

ADO adds an **Error** object to the **Errors** collection for each provider error or warning it receives. An application can enumerate the **Errors** collection to trace the errors that the provider passes.

EditMode Property (ADO)

The **EditMode** property on a **Recordset** object indicates the editing status of the current record. This property returns a Long value representing an **EditModeEnum**.

Syntax

```
currentMode = recordset.EditMode
connection.CursorType= newType
```

Remarks

The **EditMode** property is used to return the editing status of the current recordset. This property can be one of the enumerated values for **EditModeEnum** listed in the following table.

Enumeration	Value	Description
adEditNone	0	This value indicates that no editing operation is in progress.
adEditInProgress	1	This value indicates that data in the current record has been modified but not saved.
adEditAdd	2	This value indicates that the **AddNew** method has been called, and the current record in the copy buffer is a new record that has not been saved in the database.
adEditDelete	4	This value indicates that the current record has been deleted.

ActiveX® Data Objects (ADO) maintains an editing buffer associated with the current record. The **EditMode** property indicates whether changes have been made to this buffer, or whether a new record has been created. Use the **EditMode** property to determine the editing status of the current record. You can test for pending changes if an editing process has been interrupted and determine whether you need to use the Update Method or the CancelUpdate Method.

See the AddNew Method for a more detailed description of the **EditMode** property under different editing conditions.

EOF Property (ADO)

The **EOF** property on a **Recordset** object indicates that the current record position is after the last record in a **Recordset** object. This property returns a Boolean value.

Syntax

```
IsEOF = recordset.EOF
```

Remarks

The **EOF** property is used to determine whether a **Recordset** object contains records or whether you have gone beyond the limits of a **Recordset** object when you move from record to record.

The **EOF** property returns **True** if the current record position is after the last record and **False** if the current record position is on or before the last record.

If the **EOF** property is **True**, there is no current record.

If a **Recordset** object is opened containing no records, both the **BOF** and **EOF** properties are set to **True** and the **Recordset** object's **RecordCount** property setting is zero. When a **Recordset** object is opened that contains at least one record, the first record is the current record and the **BOF** and **EOF** properties are **False**.

If the last remaining record in the **Recordset** object is deleted, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

This table below indicates which **Move** methods are allowed with different combinations of the **BOF** and **EOF** properties.

	MoveFirst MoveLast	**MovePrevious Move < 0**	**Move 0**	**MoveNext Move > 0**
BOF=True EOF=False	Allowed	Error	Error	Allowed
BOF=False EOF=True	Allowed	Allowed	Error	Error
Both True	Error	Error	Error	Error
Both False	Allowed	Allowed	Allowed	Allowed

Allowing a **Move** method does not guarantee that the method will successfully locate a record; it only means that calling the specified **Move** method will not generate an error.

The following table shows what happens to the **BOF** and **EOF** property settings when various **Move** methods are called, but are unable to successfully locate a record.

	BOF Property	**EOF Property**
MoveFirst MoveLast	Set to **True**	Set to **True**
Move 0	No change	No change
MovePrevious Move < 0	Set to **True**	No change
MoveNext Move > 0	No change	Set to **True**

Execute Method on a Command Object (ADO)

The **Execute** method on a **Command** object executes the statement specified in the **CommandText** property.

Syntax

```
command.Execute(RecordsAffected,Parameters,Options)
set recordSet = command.Execute( RecordsAffected, Parameters,
Options )
```

Parameters

RecordsAffected

This optional parameter specifies a Long variable to which the provider returns the number of records that the operation affected.

Parameters

This optional parameter specifies a Variant array of parameter values passed with an SQL statement and is not used by the OLE DB Provider for AS/400 and VSAM.

Options

This optional parameter specifies a Long value representing a **CommandTypeEnum** value that indicates how the provider should evaluate the **CommandText** property of the **Command** object.

The **CommandTypeEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adCmdUnspecified	-1	This value indicates that the **CommandText** property is unspecified. This value is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM.
adCmdText	1	This value evaluates the **CommandText** property as a as a textual definition of a command or stored procedure call.
adCmdTable	2	This value evaluates the **CommandText** property as a table name. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adCmdStoredProc	4	This value evaluates the **CommandText** property as a stored procedure name. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adCmdUnknown	8	This value indicates that the type of command in **CommandText** is not known. This value is the default. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

The default for *Option* is a**dCmdText** under the OLE DB Provider for AS/400 and VSAM and the OLE DB Provider for DB2.

Remarks

When used with the OLE DB Provider for AS/400 and VSAM, the **Execute** method on a **Command** object can be used to open tables or execute DDM commands on a remote DDM server. The *Options* parameter must be set to **adCmdText** for use with the OLE DB Provider for AS/400 and VSAM.

The primary purpose of the **Command** object in the context of the OLE DB Provider for AS/400 and VSAM is to issue AS/400 Command Language (CL) commands for execution by the remote OS/400 DDM target server. To invoke DDM commands on a remote DDM server, the **CommandText** property defines the text version of a command which must have been set to:

```
EXEC COMMAND DDMCmd
```

where *DDMCmd* represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands allow you to request functions from the OS/400 operating system. Some examples are the DLTF (Delete File) or DSPFFD (Display File Description) commands. These are the same commands that could be issued on the command line if you were connected to an AS/400 via a 5250 terminal session. See the 'OS/400 CL Reference for your platform for a detailed list of possible commands.

Note that this method does not return the results or output of a remote DDM CL command. If the results or output of a remote command are to be captured, the ***DDMCmd*** statement to be executed must include syntax to redirect the command output to a file on the AS/400 host and then explicitly open this output file after the command has completed.

The **Execute** method on a **Command** object can also be used to open a data file after a **Connection** object has been opened and the **ActiveConnection** property has been set to this open connection. The **CommandText** property defines the data file to open and must be set to:

```
EXEC OPEN DataSetName
```

where *DataSetName* represents a valid data file or library member on the host. When used in this way, the **Execute** method returns a **Recordset** object. If you open a host data file from a **Command** object, then the data file is opened as read-only. This results from the limitation that no argument or option is passed by ActiveX® Data Objects (ADO) that supplies a parameter describing whether the data set should be opened as read-only or updatable.

When used with the OLE DB Provider for DB2, the **Execute** method on a **Command** object can be used to execute SQL statements or call a stored procedure. The **CommandText** property defines the SQL statements to execute and must be set to one of the following:

```
EXEC SQLStatement
```

where SQLStatement represents a valid SQL statement supported by DB2.

```
CALL StoredProcedure
```

where StoredProcedure represents a valid DB2 stored procedure on the database server.

If errors occur, these can be examined with the **Errors** collection on the **Command** object.

Execute Method on a Connection Object (ADO)

The **Execute** method on a **Connection** object executes the statement specified in the **CommandText** property.

Syntax

```
connection.Execute CommandText, RecordsAffected,Options
set recordset = connection.Execute( CommandText, RecordsAffected,
Options)
```

Parameters

RecordsAffected

This optional parameter specifies a Long variable to which the provider returns the number of records that the operation affected.

Options

This optional parameter specifies a Long value representing a **CommandTypeEnum** value that indicates how the provider should evaluate the **CommandText** property of the Command object.

The **CommandTypeEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adCmdUnspecified	-1	This value indicates that the **CommandText** property is unspecified. This value is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adCmdText	1	This value evaluates the **CommandText** property as a textual definition of a command or stored procedure call.
adCmdTable	2	This value evaluates the **CommandText** property as a table name. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adCmdStoredProc	4	This value evaluates the **CommandText** property as a stored procedure name. This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adCmdUnknown	8	This value indicates that the type of command in **CommandText** is not known. This value is the default. This value is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

Remarks

The **Execute** method on a **Connection** object can be used to open tables or execute DDM commands on a remote DDM server. The *Options* parameter must be set to **adCmdText** for use with the OLE DB Provider for AS/400 and VSAM.

The **Execute** method on a **Connection** object is primarily used to open a data file after a **Connection** object has been created. The **CommandText** property defines the data file to open and must be set to:

```
EXEC OPEN DataSetName
```

where *DataSetName* represents a valid data file or library member on the host. When used in this way, the **Execute** method returns a **Recordset** object. If you open a host data file from a **Connection** object, then the data file is opened as read-only. This results from the limitation that no argument or option is passed by ActiveX® Data Objects (ADO) that supplies a parameter describing whether the data set should be opened as read-only or updatable. If a **Recordset** object is desired that is

not read-only, then first create a **Recordset** object with the desired property settings, and use the **Recordset** objects **Open** method to return the open **Recordset**.

The **Execute** method on a **Connection** object can also be used to issue AS/400 Command Language (CL) commands for execution by the remote OS/400 DDM target server. To invoke DDM commands on a remote DDM server, the **CommandText** property defines the text version of a command which must have been set to:

```
EXEC COMMAND DDMCmd
```

where *DDMCmd* represents a valid OS/400 control language (CL) command. Note that only OS/400 CL commands are supported. These commands allow you to request functions from the OS/400 operating system. Some examples are the DLTF (Delete File) or DSPFFD (Display File Description) commands. These are the same commands that could be issued on the command line if you were connected to an AS/400 via a 5250 terminal session. See the 'OS/400 CL Reference for your platform for a detailed list of possible commands.

Note that this method does not return the results or output of a remote DDM CL command. If the results or output of a remote command are to be captured, the ***DDMCmd*** statement to be executed must include syntax to redirect the command output to a file on the AS/400 host and then explicitly open this output file after the command has completed.

If errors occur, these can be examined with the **Errors** collection on the **Connection** object.

Filter Property (ADO)

The **Filter** property on a **Recordset** object indicates a filter for data in a **Recordset**. This property sets or returns a Variant value.

Syntax

```
recordset.Filter=Criteria
```

Parameters

Criteria

This parameter specifies a Variant.

It can be either a criteria (a where clause) or one of the enumerated values for **FilterGroupEnum** listed in the following table.

Enumeration	Value	Description
adFilterNone	0	No filter. This value removes the current filter and restores all records to view.
adFilterPendingRecords	1	Use the pending records. This value allows viewing only those records that have changed but have not yet been sent to the server. This value is only applicable for batch update mode.
adFilterAffectedRecords	2	Use only records affected by the last **Delete**, **Resync**, **UpdateBatch**, or **CancelBatch** call.
adFilterFetchedRecords	3	Use the last fetched records. This value allows viewing the records in the current cache returned as a result of the last call to retrieve records (implying a resynchronization).
adFilterConflictingRecords	5	Use the conflicting records. This value allows viewing only those records that failed the last batch update.

Remarks

The **Filter** property is not supported by the Microsoft® OLE DB Provider for DB2 or the Microsoft ODBC Driver for DB2.

The **Filter** property is supported by the Microsoft OLE DB Provider for AS/400 and VSAM on certain files. In order to use the Recordset **Filter** property, an AS/400 logical file, an AS/400 keyed physical file, a mainframe KSDS file with a unique key, or a mainframe RRDS file with a unique key must be used. If this property is used on an AS/400 non-keyed physical file or any other mainframe file type, then the method fails.

When the **Filter** property is used with a criteria, the where clause is a combination of triplets. Each triplet consists of a column name, an operator, and a literal value. These where clause triplets can be combined with **AND**s and **OR**s for more complex logical filters.

If *Criteria* is a single-condition where clause, then any operator can be used. The construction of a single-condition where clause consists of a column name (the database field), an operator (greater than or equal, for example), and a literal value.

Examples of a single-condition where clause is as follows:

- recordset.Filter = "LastName = 'Jones' "
- recordset.Filter ="Salary > 30000.0"

The *Criteria* argument can be a two-condition with the following restrictions:

- If the column name (the database field) is the same in both clauses, then the separate where clauses must define a contiguous range.

- If the column names are different, then the operators must be the same. If the operators are "LIKE", then the filtered region may be unexpected.

Examples of acceptable two-condition where clauses are as follows:

- recordset.Filter = "LastName = 'Jones' **AND** FirstName = 'Tom' "
- recordset.Filter = "Cost = 5000.00 **OR** Cost > 5000.00 "

The *Criteria* argument can be three or more conditions with the following restrictions:

- The operators must be the same for all conditions. If the operators are "LIKE", then the filtered region may be unexpected.

In all cases, if the "=" operator is used, then the column names specified in the where clause must be keyed columns in the file.

One restriction on these combinations is that **OR** clauses can only be used at the highest (major) level of the logical operation.

Examples of acceptable *Criteria* meeting these conditions are:

- recordset.Filter = "(Title='Manager' **AND** Salary>30000) **OR** (Title='Administrator' **AND** Salary>50000)"
- recordset.Filter ="Salary > 30000.0"

An example of illegal *Criteria* is:

- recordset.Filter = "(Title='Associate' **OR** Salary >30000) **AND** (Title='Administrator')"

The operator can also use wildcards (* or %) in character expressions as follows:

- recordset.Filter = "Lastname LIKE '*SMITH*'"

To determine if any records were found meeting the *Criteria*, the application should check the **Recordset EOF** property. If **EOF** is true, then no records were found meeting the where clause specified in the *Criteria* parameter.

If the CursorLocation Property is set to **adUseClient** (use the client cursor engine), the **Filter** property will work if MDAC 2.0 or later is installed but will not work properly with earlier versions of ADO.

When operating on large VSAM files and only querying data on a subset of the records, using the **Filter** property is not desirable because of the performance impact. The entire VSAM file is transferred to the client for filtering. A better solution is to use the server cursor engine and the **Find** method.

Find Method (ADO)

The **Find** method on a **Recordset** object locates or seeks to the next record in a **Recordset** object that meets a particular condition and makes this the current record. This method can be used to seek to a specific record in a **Recordset** object based on a where clause (similar to an SQL where clause) defined by the user.

Syntax

```
recordset.FindCriteria, SkipRecords, SearchDirection, Start
```

Parameters

ActiveX® Data Objects (ADO) supports four arguments for the **Find** method, but the last three arguments are optional and have default values as noted below:

Criteria

This BSTR parameter specifies the criteria used for locating or seeking to a record in a **Recordset** object.

SkipRecords

This optional parameter specifies a Long expression that indicates the number of records to skip (whether to skip the current record) when locating a record in a **Recordset** object. The default value for this argument is 0 (do not skip the current record). The first time a **Find** method is used, this argument is usually set to 0 (the default). On subsequent calls to this method to seek other records that meet the specified condition, this argument would normally be set to 1, to skip one record forward before finding the next record that matches the search *Criteria*. A negative value for this parameter is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM.

SearchDirection

This optional parameter is an enumeration that specifies the direction for the search.

It can be one of the enumerated values for **SearchDirectionEnum** listed in the following table.

Enumeration	Value	Description
adSearchForward	0	Search forward from the current record.
adSearchBackward	1	Search backward from the current record. This option is not supported by the OLE DB Provider for AS/400 and VSAM.

This optional argument defaults to **adSearchForward**.

Start

This optional parameter specifies the starting location for a search, which can be a bookmark or an enumeration indicating the current, first, or last record in a **Recordset** object.

This argument is a Variant and can be either a bookmark or one of the enumerated values for **BookmarkEnum** listed in the following table.

Enumeration	Value	Description
adBookmarkCurrent	0	The current record.
adBookmarkFirst	1	The first record.
adBookmarkLast	2	The last record.

This optional argument defaults to **adBookmarkCurrent**.

Remarks

The **Find** method is not supported by the Microsoft OLE DB Provider for DB2 or the Microsoft ODBC Driver for DB2.

The **Find** method is supported by the Microsoft OLE DB Provider for AS/400 and VSAM on certain files. In order to use the Recordset **Find** method, an AS/400 logical file, an AS/400 keyed physical file, a mainframe KSDS file with a unique key, or a mainframe RRDS file with a unique key must be used. If this method is used on an AS/400 non-keyed physical file or any other mainframe file type, then the method fails.

The first parameter is the only required argument for the **Find** method. All of the other arguments are optional and have default values. This first argument is a single-condition where clause. The construction of a single-condition where clause consists of a column name (the database field), an operator (greater than or equal, for example), and a literal value.

Examples of acceptable single-condition where clauses are as follows:

- recordset.Find, "Cost > 10000.00"
- recordset.Find, "Cost < 100.00"
- recordset.Find, "Cost = 5000.00"
- recordset.Find, "LastName = 'Jones' "

Note that variables cannot be used as substitutes for literal values. If the file has multiple keys in the index, using the "=" operator will always fail since the values of all keys cannot be specified.

If the CursorLocation Property is set to **adUseClient** (use the client cursor engine), the **Filter** method will work if MDAC 2.0 or later is installed, but will not work properly with earlier versions of ADO.

When operating on large VSAM files and only querying data on a subset of the records, using the **Filter** property is not desirable because of the performance impact. The entire VSAM file is transferred to the client for filtering. A better solution is to use the server cursor engine and the **Find** method.

GetChunk Method (ADO)

The **GetChunk** method on a **Field** object returns all or a portion of the contents of a large text or binary data **Field** object.

Syntax

```
variable = field.GetChunk(Size)
```

Parameters

Size

This parameter specifies a Long expression equal to the number of bytes or characters to be retrieved.

Return Value

A Variant.

Remarks

The **GetChunk** method on a **Field** object is used to retrieve part or all of its long binary or character data. In situations where system memory is limited, the **GetChunk** method can be used to manipulate long values in portions rather than in their entirety.

The data that a **GetChunk** method returns is assigned to a variable. If the *Size* parameter is greater than the remaining data, the **GetChunk** method returns only the remaining data without padding the variable with empty spaces. If the **Field** object is empty, the **GetChunk** method returns Null.

Each subsequent **GetChunk** method call retrieves data starting from where the previous **GetChunk** call left off. However, if you are retrieving data from one field and then set or read the value of another field in the current record, ActiveX® Data Objects (ADO) assumes you are finished retrieving data from the first field. If the **GetChunk** method is called on the first field again, ADO interprets the call as a new **GetChunk** operation and starts reading from the beginning of the data. Accessing **Field** objects in other **Recordset** objects (that are not clones of the first **Recordset** object) will not disrupt **GetChunk** operations.

If the **adFldLong** bit in the **Attributes** property of a **Field** object is set to **True**, the **GetChunk** method can be used for that **Field** object.

If there is no current record when the **GetChunk** method is invoked on a **Field** object, error 3021 (no current record) occurs.

GetRows Method (ADO)

The **GetRows** method on a **Recordset** object retrieves multiple records of a recordset into an array.

Syntax

```
array = recordset.GetRows(Rows,Start,Fields)
```

Parameters

Rows

This optional parameter specifies a Long expression indicating the number of records to retrieve. The default value if this parameter is not specified is a **GetRowSetEnum** which is **adGetRowsRest** (value = -1).

Start

This optional parameter specifies the starting location for the record from which the **GetRows** operation should begin, which can be a bookmark or an enumeration indicating the current, first, or last record in a **Recordset** object.

This argument is a Variant and can be either a bookmark or one of the enumerated values for **BookmarkEnum** listed in the following table.

Enumeration	Value	Description
adBookmarkCurrent	0	The current record
adBookmarkFirst	1	The first record
adBookmarkLast	2	The last record

This optional argument defaults to **adBookmarkCurrent**.

Fields

This optional parameter is a Variant and specifies a single field name or ordinal position or an array of field names or ordinal position numbers. ADO returns only the data in these fields.

Return Value

A two-dimensional array.

Remarks

The **GetRows** method is used to copy records from a recordset into a two-dimensional array. The first subscript of the array identifies the field and the second array subscript identifies the record number. The array variable is automatically dimensioned to the correct size when the **GetRows** method returns the data.

If a value is not specified for the *Rows* parameter, the **GetRows** method automatically retrieves all the records in the **Recordset** object. If more records are requested than are available, **GetRows** returns only the number of available records.

If the **Recordset** object supports bookmarks, you can specify at which record the **GetRows** method should begin retrieving data by passing the value of that record's **Bookmark** property.

To restrict the fields that the **GetRows** method returns, you can pass either a single field name/number or an array of field names/numbers in the *Fields* argument.

After the **GetRows** method is called, the next unread record becomes the current record, or the **EOF** property is set to **True** if there are no more records.

IsolationLevel Property (ADO)

The **IsolationLevel** property on a **Connection** object indicates the level of isolation for a **Connection** object. This property sets or returns a Long value representing an **IsolationLevelEnum**. The default value for this property is **adXactChaos.**

Syntax

```
currentIsoLevel = connection.IsolationLevel
connection.IsolationLevel= newIsoLevel
```

Remarks

The **IsolationLevel** property is used to set the type of isolation level placed on a connection that the provider should use when opening the **Connection** object. This property can also be used to return the type of isolation level in use on an open **Connection** object.

This property can be one of the enumerated values for **IsolationLevelEnum** listed in the following table.

Enumeration	Value	Description
adXactUnspecified	-1	This value indicates that the provider is using a different isolation level than specified, but that the level cannot be determined.
adXactChaos	16	This value indicates that pending changes from more highly isolated transactions cannot be overwritten.
adXactBrowse	256	This value indicates that from one transaction you can view uncommitted changes in other transactions.
adXactReadUncommitted	256	Same as **adXactBrowse**.
adXactCursorStability	4096	This value indicates that from one transaction you can view changes in other transactions only after they have been committed.
adXactReadCommitted	4096	Same as **adXactCursorStability**.
adXactRepeatableRead	65536	This value indicates that from one transaction you cannot see changes made in other transactions, but that re-querying can retrieve new **Recordset** objects.
adXactIsolated	1048576	This value indicates that transactions are conducted in isolation of other transactions.
adXactSerializable	1048576	Same as **adXactIsolated**.

This property is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM.

When setting the **IsolationLevel** property, this change does not take effect until the next time that the **BeginTrans** method is called.

If the level of isolation you request is unavailable, the provider may return the next greater level of isolation.

When used with the Microsoft OLE DB Provider for DB2 or the Microsoft OLE DB Driver for DB2, the ActiveX® Data Objects (ADO) **IsolationLevel** property corresponds with the isolation level defined by the ANSI SQL standard and with IBM's DB2 implementation of isolation level. The table below indicates how the ADO **IsolationLevel** property corresponds with the terms used by ANSI SQL for isolation level and with IBM documentation for isolation level in DB2.

ADO IsolationLevel property	ANSI SQL isolation level	IBM documentation
	AUTOCOMMITTED (Note that this applies only to DB2/400 and does not correspond with an ANSI SQL isolation level	COMMIT(*NONE) (NC). This isolation level is used in DB2/400 auto-commit mode only and has no corresponding isolation level on other DB2 platforms or in ANSI SQL.
adXactReadUncommitted	READ UNCOMMITTED	UNCOMMITTED READ (UR). This isolation level corresponds with ANSI SQL READ UNCOMMITTED.
adXactReadCommitted or **adXactCursorStability**	READ COMMITTED	CURSOR STABILITY (CS). This isolation level corresponds with ANSI SQL READ COMMITTED.
adXactRepeatableRead	REPEATABLE READ	READ STABILITY (RS). This isolation level corresponds with ANSI SQL REPEATABLE READ.
adXactSerializable or **adXactIsolated**	SERIALIZABLE	REPEATABLE READ (RR). This isolation level corresponds with ANSI SQL SERIALIZABLE.

When used with the Remote Data Service on a client-side **Connection** object, the **IsolationLevel** property can be set only to **adXactUnspecified**. Because users are working with disconnected **Recordset** objects on a client-side cache, there may be multi-user issues. For instance, when two different users try to update the same record, Remote Data Service simply allows the user who updates the record first to "win." The second user's update request will fail with an error.

Item Method (ADO)

The **Item** method on a **Collection** object returns a specific member object of a collection by name or ordinal number. This method is supported for the **Errors**, **Fields**, and **Properties** collections using the OLE DB Provider for AS/400 and VSAM.

Syntax

```
set Object = collection.Item ( Index )
```

Parameters

Index

This parameter specifies a Variant that evaluates either to the name or to the ordinal number of an object in a collection.

Return Value

An object reference from the collection.

Remarks

The **Item** method is used to return a specific object in a collection. If the method cannot find an object in the collection corresponding to the *Index* parameter, an error occurs. Also, some collections do not support named objects; for these collections, you must use ordinal number references.

The **Item** method is the default method for all collections; therefore, the following syntax forms are interchangeable:

collection.Item (*Index*)

collection (*Index*)

This method is only supported on the **Errors**, **Fields**, and **Properties** collections under the OLE DB Provider for AS/400 and VSAM.

LockType Property (ADO)

The **LockType** property on a **Recordset** object indicates the type of locks placed on records during editing. This property sets or returns a Long value representing a **LockTypeEnum**. The default value for this property is **adLockReadOnly**.

Syntax

```
currentLock = recordset.LockType
recordset.LockType= newtype
```

Remarks

The **LockType** property is used to set the type of locks placed on records that the provider should use when opening the **Recordset**. This property can also be used to return the type of locking in use on an open **Recordset** object. This property can be one of the enumerated values for **LockTypeEnum** as listed in the following table.

Enumeration	Value	Description
adLockUnspecified	-1	This value does not specify a type of lock. For a recordset created with the **Clone** method, the clone is created with the same lock type as the original.
adLockReadOnly	1	This value indicates read-only records where the data cannot be altered.
adLockPessimistic	2	This value indicates pessimistic locking, record by record. The provider does what is necessary to ensure successful editing of the records, usually by locking records at the data source immediately after editing. This lock type is supported by the Microsoft® OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2. However, the OLE DB Provider for AS/400 and VSAM internally maps this lock type to **adLockBatchOptimistic**.
adLockOptimistic	3	This value indicates optimistic locking, record by record. The provider uses optimistic locking, locking records only when the **Update** method is called. This lock type is not supported by the OLE DB Provider for DB2.
adLockBatchOptimistic	4	This value indicates optimistic batch updates and is required for batch update mode. This option is not supported by the OLE DB Provider for DB2.

If a provider cannot support the requested **LockType** setting, it will substitute another type of locking. To determine the actual locking options available on a **Recordset** object, use the **Supports** method with **adUpdate** and **adUpdateBatch**.

The **adLockPessimistic** setting is not supported if the **CursorLocation** property is set to **adUseClient**. If an unsupported value is set, then no error will result; the closest supported **LockType** will be used instead.

The **LockType** property is read/write when the **Recordset** is closed and read-only when it is open.

MaxRecords Property (ADO)

The **MaxRecords** property on a **Recordset** indicates the maximum number of records to return to a **Recordset** from a query. This property sets or returns a Long value that indicates the maximum number of records to return. The default value for this property is zero (no limit).

Syntax

```
cntRecords = recordset.MaxRecords
recordset.MaxRecords = count
```

Remarks

The **MaxRecords** property is used to limit the number of records returned from a data source. The default setting of this property is zero, which means that all requested records are returned.

The **MaxRecords** property is read/write when the **Recordset** is closed and read-only when it is open.

Mode Property (ADO)

The **Mode** property on a **Connection** object sets or returns the available permissions for modifying data on a **Connection**.

Syntax

```
currentMode = connection.Mode
connection.Mode = newMode
```

Remarks

The **Mode** property is used to set or return the access permissions in use by the provider on the current connection. The **Mode** property can only be set when the **Connection** object is closed.

Several of the ActiveX® Data Objects (ADO) enumerated values for the mode settings imply that the using certain mode settings will prevent other applications from opening a connection to the file or table. Under the Microsoft® OLE DB provider for AS/400 and VSAM, these mode settings result in a file lock, but do not prevent other applications from opening a connection.

The value for the **Mode** can be one of the enumerated values for **ConnectModeEnum** as listed in the following table.

Enumeration	**Value**	**Description**
adModeUnknown	0	This value indicates that the permissions have not yet been set or cannot be determined.
adModeRead	1	This value indicates read-only permissions.
adModeWrite	2	This value indicates write-only permissions. This value is not supported by the OLE DB provider for AS/400 and VSAM.
adModeReadWrite	3	This value indicates read/write permissions.
adModeShareDenyRead	4	This value prevents others from opening a connection with read permissions.
adModeShareDenyWrite	8	This value prevents others from opening a connection with write permissions. Under the OLE DB provider for AS/400 and VSAM, this mode setting is implemented as a file lock and does not result in excluding other applications from opening a connection. Other applications opening a connection will not receive an error, but the table or file will be locked preventing any changes from other applications.
adModeShareExclusive	0xc	This value prevents others from opening a connection. Under the OLE DB provider for AS/400 and VSAM, this mode setting is implemented as a file lock and does not result in excluding other applications from opening a connection. Other applications opening a connection will not receive an error, but the table or file will be locked preventing any changes from other applications.

adModeShareDenyNone	0x10	This value prevents others from opening a connection with any permissions. Under the OLE DB provider for AS/400 and VSAM this mode setting is implemented as a file lock and does not result in excluding other applications from opening a connection. Other applications opening a connection will not receive an error, but the table or file will be locked preventing any changes from other applications.

The **Mode** property defaults to **adModeUnknown**.

Move Method (ADO)

The **Move** method on a **Recordset** object moves the position of the current record in a **Recordset** object.

Syntax

```
recordset.Move NumRecords, Start
```

Parameters

NumRecords

This parameter specifies a signed Long expression specifying the number of records the current record position moves.

Start

This optional parameter specifies the starting location for the record from which the **Move** operation should begin, which can be a bookmark or an enumeration indicating the current, first, or last record in a **Recordset** object.

This argument is a Variant and can be either a bookmark or one of the enumerated values for **BookmarkEnum** as listed in the following table.

Enumeration	Value	Description
adBookmarkCurrent	0	The current record
adBookmarkFirst	1	The first record
adBookmarkLast	2	The last record

This optional argument defaults to **adBookmarkCurrent**.

Return Value

None.

Remarks

The **Move** method is supported on all **Recordset** objects.

If the *NumRecords* parameter is greater than zero, the current record position moves forward (toward the end of the recordset). If *NumRecords* is less than zero, the current record position moves backward (toward the beginning of the recordset).

If the **Move** method would move the current record position to a point before the first record, ActiveX® Data Objects (ADO) sets the current record to the position before the first record in the recordset (the **BOF** property is set to **True**). An attempt to move backward when the **BOF** property is already **True** generates an error.

If the **Move** call would move the current record position to a point after the last record, ADO sets the current record to the position after the last record in the recordset (the **EOF** property is set to **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

Invoking the **Move** method from an empty **Recordset** object generates an error.

If the *Start* parameter is specified, the move is relative to the record with this bookmark assuming the **Recordset** object supports bookmarks. If not specified, the move is relative to the current record.

If the **CacheSize** property is set to greater than 1 to locally cache records from the provider, passing a *NumRecords* value that moves the current record position outside of the current group of cached records forces ADO to retrieve a new group of records starting from the destination record. The **CacheSize** property determines the size of the newly retrieved group, and the destination record is the first record retrieved.

If the **Recordset** object is forward-only, a user can still pass a *NumRecords* value less than zero as long as the destination is within the current set of cached records. If the **Move** method call would move the current record position to a record before the first cached record, an error occurs. Thus, you can use a record cache that supports full scrolling over a provider that only supports forward scrolling. Because cached records are loaded into memory, you should avoid caching more records than necessary. Even if a forward-only **Recordset** object supports backward moves in this way, calling the **MovePrevious** method

on any forward-only **Recordset** object still generates an error.

MoveFirst Method (ADO)

The **MoveFirst** method on a **Recordset** object moves to the first record in a specified **Recordset** object and makes that record current.

Syntax

```
recordset.MoveFirst
```

Parameters

None.

Remarks

The **MoveFirst** method is used to move the current record position to the first record in the recordset. The **MoveFirst** method can be invoked on a forward-only **Recordset** object; but doing so may cause the provider to re-execute the command that generated the **Recordset** object.

MoveLast Method (ADO)

The **MoveLast** method on a **Recordset** object moves to the last record in a specified **Recordset** object and makes that record current.

Syntax

```
recordset.MoveLast
```

Parameters

None.

Remarks

The **MoveLast** method is used to move the current record position to the last record in the recordset.

When using a server-side cursor, the **Recordset** object must support bookmarks or backward cursor movement; otherwise, the **MoveLast** method call generates an error.

MoveNext Method (ADO)

The **MoveNext** method on a **Recordset** object moves to the next record in a specified **Recordset** object and makes that record current.

Syntax

```
recordset.MoveLast
```

Parameters

None.

Remarks

The **MoveNext** method is used to move the current record position one record forward (toward the bottom of the recordset). If the last record is the current record and the **MoveNext** method is invoked, ActiveX® Data Objects (ADO) sets the current record to the position after the last record in the recordset (the **EOF** property is set to **True**). An attempt to move forward when the **EOF** property is already **True** generates an error.

MovePrevious Method (ADO)

The **MovePrevious** method on a **Recordset** object moves to the previous record in a specified **Recordset** object and makes that record current.

Syntax

```
recordset.MovePrevious
```

Parameters

None.

Remarks

The **MovePrevious** method is used to move the current record position one record backward (toward the top of the recordset).

When using a server-side cursor, if the **Recordset** object does not support either bookmarks or backward cursor movement, the **MovePrevious** method generates an error.

If the first record is the current record and the **MovePrevious** method is invoked, ADO sets the current record to the position before the first record in the recordset (the **BOF** property is set to **True**). An attempt to move backward when the **BOF** property is already **True** generates an error.

Note that if a **MovePrevious** method is invoked after **Delete** method is invoked, this moves the current record back two records instead of one.

Name Property (ADO)

The **Name** property on a **Command** object sets or returns a string value indicating the name of the object. The **Name** property on a **Field** or **Property** returns a string value indicating the name of the object.

Syntax

```
currentName = command.Name
command.Name = newName
```

Remarks

The **Name** property is used to assign a name to or retrieve the name of a **Command**, **Field**, or **Property** object. This value is read/write on a **Command** object and read-only on a **Property** or **Field** object. Note that the **Name** property on a **Command** object cannot be assigned (set) using the OLE DB Provider for AS/400 and VSAM.

The **Name** property of an object can be retrieved by an ordinal reference, after which you can refer to the object directly by name. For example, if **recordset.Properties(20).Name** yields updatability, you can subsequently refer to this property as **recordset.Properties("Updatability")**.

NativeError Property (ADO)

The **NativeError** property on an **Error** object indicates the provider-specific error code for a given **Error** object. This property returns a Long value that indicates the error code.

Syntax

```
errorCode = currentError.NativeError
```

Remarks

The **NativeError** property on an **Error** object is used to retrieve the database-specific error information for a particular **Error** object. For example, when using the Microsoft® OLE DB Provider for DB2, native error codes that originate from the OLE DB Provider for DB2 pass through ActiveX® Data Objects (ADO) to the ADO **NativeError** property.

Number Property (ADO)

The **Number** property on an **Error** object indicates the number that uniquely identifies an **Error** object. This property returns a Long value that may correspond to one of the **ErrorValueEnum** constants.

Syntax

```
errorNumber = currentError.Number
```

Remarks

The **Number** property on an **Error** object is used to determine which error occurred. The value of the property is a unique number that corresponds to the error condition.

The **Errors** collection returns an HRESULT. These HRESULTs can be raised by underlying components, such as OLE DB or even OLE itself.

The value for the **Number** property on the **Error** object representing an ADO error (not an OLE DB error) can be one of the following enumerated values for **ErrorValueEnum** listed in the table below. Note that three forms of the error number value are listed in the table:

- Positive decimal—The lower two bytes of the full number in decimal format. This number is displayed in the default Microsoft® Visual Basic® error message dialog box. For example, **Run-time error '3707'**.
- Negative decimal—The decimal translation of the full error number.
- Hexadecimal—The hexadecimal representation of the full error number. The Windows facility code is the lowest hexadecimal fourth digit in the upper two bytes of the number. The facility code for ADO error numbers is *A*. For example: 0x800**A**0E7B.

Note that OLE DB errors may be passed to an ADO application. Typically, these OLE DB errors can be identified by a Windows® facility code of *4*. For example, 0x800**4**xxxx. For more information about the possible error numbers returned by OLE DB, see Chapter 16 of the *OLE DB Programmer's Reference*.

Enumeration	Value	Description
adErrBoundToCommand	3707 -2146824581 0x800A0E7B	The application cannot change the **ActiveConnection** property of a **Recordset** object that has a **Command** object as its source.
adErrCannotComplete	3732 -2146824556 0x800A0E94	The server cannot complete the operation.
adErrCantChangeConnection	3748 -2146824540 0x800A0EA4	The connection was denied. The new connection requested has different characteristics than the one already in use.
adErrCantChangeProvider	3220 -2146825068 0X800A0C94	The supplied provider is different from the one already in use.
adErrCantConvertvalue	3724 -2146824564 0x800A0E8C	The data value cannot be converted for reasons other than sign mismatch or data overflow. For example, conversion would have truncated data.
adErrCantCreate	3725 -2146824563 0x800A0E8D	The data value cannot be set or retrieved because the field data type was unknown, or the provider had insufficient resources to perform the operation.
adErrCatalogNotSet	3747 -2146824541 0x800A0EA3	The operation requires a valid **ParentCatalog**.

adErrColumnNotOnThisRow	3726 -2146824562 0x800A0E8E	The record does not contain this field.
adErrDataConversion	3421 -2146824867 0x800A0D5D	The application uses a value of the wrong type for the current operation.
adErrDataOverflow	3721 -2146824567 0x800A0E89	The data value is too large to be represented by the field data type.
adErrDelResOutOfScope	3738 -2146824550 0x800A0E9A	The URL of the object to be deleted is outside the scope of the current record.
adErrDenyNotSupported	3750 -2146824538 0x800A0EA6	The provider does not support sharing restrictions.
adErrDenyTypeNotSupported	3751 -2146824537 0x800A0EA7	The provider does not support the requested kind of sharing restriction.
adErrFeatureNotAvailable	3251 -2146825037 0x800A0CB3	The object or provider is not capable of performing requested operation.
adErrFieldsUpdateFailed	3749 -2146824539 0x800A0EA5	The **Fields** update failed. For further information, examine the **Status** property of individual field objects.
adErrIllegalOperation	3219 -2146825069 0x800A0C93	The operation is not allowed in this context.
adErrIntegrityViolation	3719 -2146824569 0x800A0E87	The data value conflicts with the integrity constraints of the field.
adErrInTransaction	3246 -2146825042 0x800A0CAE	The **Connection** object cannot be explicitly closed while in a transaction.
adErrInvalidArgument	3001 -2146825287 0x800A0BB9	The arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.
adErrInvalidConnection	3709 -2146824579 0x800A0E7D	The operation is not allowed on an object referencing a closed or invalid connection.
adErrInvalidParamInfo	3708 -2146824580 0x800A0E7C	The **Parameter** object is improperly defined. Inconsistent or incomplete information was provided.
adErrInvalidTransaction	3714 -2146824574 0x800A0E82	The coordinating transaction is invalid or has not started.
adErrInvalidURL	3729 -2146824559 0x800A0E91	The URL contains invalid characters. Make sure the URL is typed correctly.
adErrItemNotFound	3265 -2146825023 0x800A0CC1	The item cannot be found in the collection corresponding to the requested name or ordinal.
adErrNoCurrentRecord	3021 -2146825267 0x800A0BCD	Either the **BOF** or **EOF** property is True, or the current record has been deleted. The requested operation requires a current record.

adErrNotExecuting	3715 -2146824573 0x800A0E83	The operation cannot be performed while not executing.
adErrNotReentrant	3710 -2146824578 0x800A0E7E	The operation cannot be performed while processing event.
adErrObjectClosed	3704 -2146824584 0x800A0E78	The operation is not allowed when the object is closed.
adErrObjectInCollection	3367 -2146824921 0x800A0D27	The object is already in collection. Cannot append.
adErrObjectNotSet	3420 -2146824868 0x800A0D5C	The object is no longer valid.
adErrObjectOpen	3705 -2146824583 0x800A0E79	The operation is not allowed when the object is open.
adErrOpeningFile	3002 -2146825286 0x800A0BBA	The file could not be opened.
adErrOperationCancelled	3712 -2146824576 0x800A0E80	The operation has been cancelled by the user.
adErrOutOfSpace	3734 -2146824554 0x800A0E96	The operation cannot be performed. Provider cannot obtain enough storage space.
adErrPermissionDenied	3720 -2146824568 0x800A0E88	Insufficient permission prevents writing to the field.
adErrPropConflicting	3742 -2146824546 0x800A0E9E	Property value conflicts with a related property.
adErrPropInvalidColumn	3739 -2146824549 0x800A0E9B	Property cannot apply to the specified field.
adErrPropInvalidOption	3740 -2146824548 0x800A0E9C	Property attribute is invalid.
adErrPropInvalidValue	3741 -2146824547 0x800A0E9D	Property value is invalid. Make sure the value is typed correctly.
adErrPropNotAllSettable	3743 -2146824545 0x800A0E9F	Property is read-only or cannot be set.
adErrPropNotSet	3744 -2146824544 0x800A0EA0	Optional property value was not set.
adErrPropNotSettable	3745 -2146824543 0x800A0EA1	Read-only property value was not set.
adErrPropNotSupported	3746 -2146824542 0x800A0EA2	Provider does not support the property.

adErrProviderFailed	3000 -2146825288 0x800A0BB8	Provider failed to perform the requested operation.
adErrProviderNotFound	3706 -2146824582 0x800A0E7A	Provider cannot be found. It may not be properly installed.
adErrReadFile	3003 -2146825285 0x800A0BBB	File could not be read.
adErrResourceExists	3731 -2146824557 0x800A0E93	Copy operation cannot be performed. Object named by destination URL already exists. Specify **adCopyOverwrite** to replace the object.
adErrResourceLocked	3730 -2146824558 0x800A0E92	Object represented by the specified URL is locked by one or more other processes. Wait until the process has finished and attempt the operation again.
adErrResourceOutOfScope	3735 -2146824553 0x800A0E97	Source or destination URL is outside the scope of the current record.
adErrSchemaViolation	3722 -2146824566 0x800A0E8A	Data value conflicts with the data type or constraints of the field.
adErrSignMismatch	3723 -2146824565 0x800A0E8B	Conversion failed because the data value was signed and the field data type used by the provider was unsigned.
adErrStillConnecting	3713 -2146824575 0x800A0E81	Operation cannot be performed while connecting asynchronously.
adErrStillExecuting	3711 -2146824577 0x800A0E7F	Operation cannot be performed while executing asynchronously.
adErrTreePermissionDenied	3728 -2146824560 0x800A0E90	Permissions are insufficient to access tree or subtree.
adErrUnavailable	3736 -2146824552 0x800A0E98	Operation failed to complete and the status is unavailable. The field may be unavailable or the operation was not attempted.
adErrUnsafeOperation	3716 -2146824572 0x800A0E84	Safety settings on this computer prohibit accessing a data source on another domain.
adErrURLDoesNotExist	3727 -2146824561 0x800A0E8F	Either the source URL or the parent of the destination URL does not exist.
adErrURLNamedRowDoesNotExist	3737 -2146824551 0x800A0E99	Record named by this URL does not exist.
adErrVolumeNotFound	3733 -2146824555 0x800A0E95	Provider cannot locate the storage device indicated by the URL. Make sure the URL is typed correctly.
adErrWriteFile	3004 -2146825284 0x800A0BBC	Write to file failed.

NumericScale Property (ADO)

The **NumericScale** property on a **Field** object indicates the scale of Numeric values in a **Field** object. This property returns a byte value indicating the number of decimal places to which numeric values will be resolved.

Syntax

```
scale = currentfield.NumericScale
```

Remarks

The **NumericScale** property is used to determine how many digits to the right of the decimal point will be used to represent values for a numeric **Field** object.

The byte value that the **NumericScale** property will return is dependent on the data type of the **Field** object. The value for the ActiveX® Data Objects (ADO) data type of the **Field** object can be one of the enumerated values for **DataTypeEnum** as listed in the following table.

Enumeration	Value	Description
adEmpty	0	This data type indicates that no value was specified (DBTYPE_EMPTY).
adSmallInt	2	This data type indicates a 2-byte (16-bit) signed integer (DBTYPE_I2).
adInteger	3	This data type indicates a 4-byte (32-bit) signed integer (DBTYPE_I4).
adSingle	4	This data type indicates a 4-byte (32-bit) single precision IEEE floating point number (DBTYPE_R4).
adDouble	5	This data type indicates an 8-byte (64-bit) double precision IEEE floating point number (DBTYPE_R8).
adCurrency	6	A data type indicates a currency value (DBTYPE_CY). Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adDate	7	This data type indicates a date value stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBSTR	8	This data type indicates a null-terminated Unicode character string (DBTYPE_BSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIDispatch	9	This data type indicates a pointer to an IDispatch interface on an OLE object (DBTYPE_IDISPATCH). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adError	10	This data type indicates a 32-bit error code (DBTYPE_ERROR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBoolean	11	This data type indicates a Boolean value (DBTYPE_BOOL). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

adVariant	12	This data type indicates an automation variant (DBTYPE_VARIANT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIUnknown	13	This data type indicates a pointer to an IUnknown interface on an OLE object (DBTYPE_IUNKNOWN). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adDecimal	14	This data type indicates numeric data with a fixed precision and scale (DBTYPE_DECIMAL).
adTinyInt	16	This data type indicates a single-byte (8-bit) signed integer (DBTYPE_I1). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedTinyInt	17	This data type indicates a single-byte (8-bit) unsigned integer (DBTYPE_UI1). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedSmallInt	18	This data type indicates a 2-byte (16-bit) unsigned integer (DBTYPE_UI2). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedInt	19	This data type indicates a 4-byte (32-bit) unsigned integer (DBTYPE_UI4). This data type is not supported by the OLE DB Provider or the OLE DB Provider for DB2.
adBigInt	20	This data type indicates an 8-byte (64-bit) signed integer (DBTYPE_I8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM.
adUnsignedBigInt	21	This data type indicates an 8-byte (64-bit) unsigned integer (DBTYPE_UI8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adGUID	72	This data type indicates a globally unique identifier or GUID (DBTYPE_GUID). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBinary	128	This data type indicates fixed-length binary data (DBTYPE_BYTES).
adChar	129	This data type indicates a character string value (DBTYPE_STR).
adWChar	130	This data type indicates a null-terminated Unicode character string (DBTYPE_WSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adNumeric	131	This data type indicates numeric data where the precision and scale are exactly as specified (DBTYPE_NUMERIC).
adUserDefined	132	This data type indicates user-defined data (DBTYPE_UDT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

adDBDate	133	This data type indicates an OLE DB date structure (DBTYPE_DATE).
adDBTime	134	This data type indicates an OLE DB time structure (DBTYPE_TIME).
adDBTimeStamp	135	This data type indicates an OLE DB timestamp structure (DBTYPE_TIMESTAMP).
adVarChar	200	This data type indicates variable-length character data (DBTYPE_STR).
adLongVarChar	201	This data type indicates a long string value.
adVarWChar	202	This data type indicates a Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adLongVarWChar	203	This data type indicates a long Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adVarBinary	204	This data type indicates variable-length binary data (DBTYPE_BYTES).
adLongVarBinary	205	This data type indicates a long binary value.

Open Method on a Connection Object (ADO)

The **Open** method on a **Connection** object opens a connection to a data source.

Syntax

```
connection.Open ConnectionString, UserID, Password
```

Parameters

ConnectionString

This optional parameter specifies a string containing connection information. See the **ConnectionString** property on a **Connection** object for details on valid settings.

UserID

This optional parameter specifies a string containing a user name to use when establishing the connection.

Password

This optional parameter specifies a string containing a password to use when establishing the connection.

Values used for the ConnectionString parameter

The information needed to establish a connection to a data source can be set in the **ConnectionString** property or passed as part of the **Open** method in the *ConnectionString* parameter. In either case, this information must be in a specific format for use with the Microsoft OLE DB Provider for AS/400 and VSAM. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons. ActiveX Data Objects (ADO) supports several standard ADO-defined arguments for the **ConnectionString** property as listed in the following table.

Argument	**Description**
Data Source	This argument specifies the name of the data source for the connection. This argument is the Data Source name stored in the registry under the OLE DB Provider for AS/400 and VSAM.
File Name	This argument specifies the name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed. This argument is not supported by the OLE DB Provider for AS/400 and VSAM.
Location	This argument specifies the Remote Database Name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.
Password	This argument specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target host system and has appropriate access rights to the file.
Provider	This argument specifies the name of the provider to use for the connection. To use the OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB". To use the Microsoft OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB". To use the Microsoft ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the **ConnectionString** since this value is the default for ADO.
Remote Provider	This argument specifies the name of a provider to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

Remote Server	This argument specifies the path name of a server to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
User ID	This argument specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target host system and has appropriate access rights to the file.

The OLE DB Provider for AS/400 and VSAM also supports a number of provider-specific arguments, some of which have default values as specified in the table below. These arguments are listed in the following table.

Argument	**Description**
BinAsChar	This parameter indicates whether to process binary fields as character fields (default is 0; do not process binary fields as character fields).
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files.
HCD FileName	The fully qualified filename of the DDM host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the SNA System directory. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400.
Local LU	The name of the local LU alias configured in Host Integration Server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
RDB	The remote database name for OS/400. You only need to specify this value if it is different from the remote LU alias configured in Host Integration Server.

Repair Host Keys	This parameter indicates whether the OLE DB provider should repair any host key values set in the registry and defaults to false.
RemoteLU	The name of the remote LU alias configured in the Host Integration server.
StrictVal	This parameter indicates whether strict validation should be used and defaults to false.

The OLE DB Provider for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by OLE DB Provider for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier OLE DB Provider for DB2 included with SNA Server 4.0.

The arguments supported by the OLE DB Provider for DB2 supplied with Host Integration Server 2009 are listed in the following table.

Argument	**Description**
BinAsChar	When this parameter is set to true, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This parameter defaults to false.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
DefSch	The name of the default schema (collection/owner) where the system catalogs resides. This parameter can be QSYS2;SYSIBM;SYSTEM; CURLIB; or USERID depending on platform. This parameter does not have a default value.
InitCat	This parameter is used as the first part of a 3-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. In DB2/400, this parameter is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then one can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE. This parameter has no default value.
LocalLU	The name of the local LU alias configured in Host Integration Server.

ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
PkgCol	The name of the DRDA target collection (AS/400 library) where the Microsoft® OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2 uses packages to issue dynamic and static SQL statements. The OLE DB Provider will create packages dynamically in the location to which the user points using the Package Collection parameter.
RemoteLU	The name of the remote LU alias configured in Host Integration Server.
TPName	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server 2009 uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPName is set to 0X07F9F9F9.

UOW	This parameter determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

The ODBC Driver for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by ODBC Driver for DB2 supplied with Host Integration Server 2009 differ from the arguments supported by the earlier ODBC Driver for DB2 included with SNA Server 4.0.

The arguments supported by the ODBC Driver for DB2 supplied with Host Integration Server 2009 are listed in the following table.

Argument	**Description**
BAC	When the *BinAsChar* parameter is set to true (1), the ODBC Driver for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The CCSID and PCCodePage values are required input parameters.
CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote computer. The CCSID property is required when processing binary data as character data. Unless the BinAsChar value is set, character data is converted based on the DB2 column CCSID and default ANSI code page. If this argument is omitted, this parameter defaults to U.S./Canada (37).
CP	The character code page to use on the computer. This parameter is required when processing binary data as character data. Unless the Binary as Character (BAC) value is set, character data is converted based on the default ANSI code page configured in Windows®. If this argument is omitted, the default value is set to Latin 1 (1252).
DESC	A field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.

DS	The Default Schema parameter is the name of the Collection where the ODBC Driver for DB2 looks for catalog information. The Default Schema is the "SCHEMA" name for the target collection of tables and views. The ODBC driver uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (e.g., ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, then the ODBC driver uses the USER_ID provided at login. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Obviously, this default is inappropriate in many cases, therefore it is essential that the Default Schema value in the data source be defined.
DSN	The data source name is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
LLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
MN	When SNA is used for the Network Transport Library (NTL), the Mode Name field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to QPCSUPP.
NA	When TCP/IP is used for the Network Transport Library (NTL), the Network Address parameter indicates the IP address or the hostname alias of the host DB2 server.
NP	When TCP/IP is used for the Network Transport Library (NTL), the Network Port parameter indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default value is TCP/IP port 446.
NTL	The Network Transport Library parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA. If the default SNA is selected, then values for LLU, MN, and RLU are required. If TCP/IP is selected, then values for NetAddr and NetPort are required.
PC	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft ODBC Driver for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. The ODBC driver will create packages dynamically in the location to which the user points using the Package Collection parameter.
PDS	The Provider Data Source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
PROV	Specifies the name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the **ConnectionString** since this value is the default for ADO.

PWD	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. Note that this parameter is the same as the Parameter parameter.
RDB	The remote database name parameter is used as the first part of a three-part, fully qualified DB2 table name. This parameter is referred to by different names depending on the DB2 platform. In DB2 on MVS and OS/390, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then a value can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE.
RLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
TPN	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPN is set to 0X07F9F9F9.
UID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is the same as the User ID parameter.
UOW	Determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note
Not all of these parameters are required. The user can also be prompted for this information.

Remarks

The **Open** method on a **Connection** object is used to open tables on a remote DDM server. Using the **Open** method on a **Connection** object establishes the physical connection to a data source. After this method successfully completes, the connection is live and other methods can be invoked on the **Connection** object to process results.

The optional *ConnectionString* parameter is used to specify a connection string containing a series of *argument=value* statements separated by semicolons. The **ConnectionString** property on a **Connection** object automatically inherits the value used for the *ConnectionString* parameter. Therefore, the **ConnectionString** property of the **Connection** object can be set before opening the **Connection** object, or the *ConnectionString* parameter can be used to set or override the current connection parameters during the **Open** method call.

If user and password information is set in both the *ConnectionString* parameter and in the optional *UserID* and *Password* parameters, the results may be unpredictable. Such information should only be passed in either the *ConnectionString* parameter or the *UserID* and *Password* parameters.

There are a number of different ways to open a connection. The **Open** method can pass all of the appropriate connection information as part of the *ConnectionString* parameter or by setting the **ConnectionString** property of the **Connection** object, if this information is known in advance. The syntax in this case using the *ConnectionString* parameter for use with the OLE DB Provider for AS/400 and VSAM is as follows:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.Open "Provider=SNAOLEDB;Data Source=REMLU;User ID=USERNAME;Password=password;Loc
al LU=LOCAL;Remote LU=DATABASE;ModeName=QPCSUPP;CCSID=37;CodePage=437"
```

Note that not all of these parameters are required. The registry settings for the Data Source usually have default values set for remote LU, local LU, APPC mode, CCSID, and CodePage. If a data source is specified, this other information is not usually needed. These registry settings are configured by using the Microsoft Management Console snap-in for the OLE DB Provider for AS/400 and VSAM.

The simplest form of an **Open** command that contains all necessary information is as follows:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.Open "Provider=SNAOLEDB;Data Source=REMLU;User ID=USERNAME;Password=password"
```

Note
The Data Source, User ID and Password must be included.

In the case where you would like the user to input the connection information, the following syntax can be used. This syntax does not specify any connection information except the provider, which is always required unless this is set in the **ConnectionString** or **Provider** property of the **Connection** object:

```
connection = CreateObject("ADODB.Connection.2.0")
connection.ConnectionString = "Provider=SNAOLEDB"
connection.Open
```

This method of invoking the **Open** method automatically causes a dialog box to appear asking the user for the data source, user name, and password.

When operations have been concluded over an open **Connection** object, the **Close** method should be invoked on the **Connection** object to free any associated system resources. Closing a **Connection** object does not remove it from memory; you may change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the **Connection** object variable to **Nothing**.

If errors occur, these can be examined with the **Errors** collection on the **Connection** object.

Open Method on a Recordset Object (ADO)

The **Open** method on a **Recordset** object opens a cursor that represents records from a base table or the results of a query.

Syntax

```
recordset.Open Source, ActiveConnection, CursorType, LockType,
Options
```

Parameters

Source

This optional parameter specifies a Variant that evaluates to a valid **Command** object variable name or a valid string specifying the command text specific to the Microsoft OLE DB Provider for AS/400 and VSAM to open a data file on the host.

ActiveConnection

This optional parameter specifies either a Variant that evaluates to a valid **Connection** object variable name or a String containing connection information equivalent to the **ConnectionString** property of a **Connection** object. Possible values are listed at the end of the **Parameters** section.

CursorType

This optional parameter specifies a **CursorTypeEnum** value that determines the type of cursor that the provider should use when opening the **Recordset**. See the CursorType Property of a **Recordset** object for more information. Possible values are listed at the end of the **Parameters** section.

LockType

This optional parameter specifies a **LockTypeEnum** value that determines what type of locking (concurrency) the provider should use when opening the recordset. See the LockType Property of a **Recordset** object for more information. Possible values are listed at the end of the **Parameters** section.

Options

This optional parameter specifies a Long value representing a **CommandTypeEnum** value that indicates how the provider should evaluate the *Source* parameter.

Values used for the ActiveConnection property

The information needed to establish a connection to a data source can be set in the **ActiveConnection** property of a **Recordset** object or passed as part of the **Open** method on a **Recordset** object in the *ActiveConnection* parameter. In either case, this information must be in a specific format for use with the OLE DB Provider for AS/400 and VSAM. This information can be a data source name (DSN) or a detailed connection string containing a series of *argument=value* statements separated by semicolons. ActiveX Data Objects (ADO) supports several standard ADO-defined arguments for the **ActiveConnection** property as listed in the following table.

Argument	**Description**
Data Source	This argument specifies the name of the data source for the connection. This argument is the optional when used with OLE DB Provider for AS/400 and VSAM or the Microsoft OLE Provider for DB2.
File Name	This argument specifies the name of the provider-specific file containing preset connection information. This argument cannot be used if a *Provider* argument is passed. This argument is not supported by the OLE DB Provider for AS/400 and VSAM.
Location	This argument specifies the Remote Database Name used for connecting to OS/400 systems. This parameter is optional when connecting to mainframe systems.

Password	This argument specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used by Host Integration Server 2009 to validate that the user can log on to the target host system and has appropriate access rights to the file.
Provider	This argument specifies the name of the provider to use for the connection. To use the OLE DB Provider for AS/400 and VSAM, the Provider string must be set to "SNAOLEDB". To use the OLE DB Provider for DB2, the Provider string must be set to "DB2OLEDB". To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the ConnectionString since this value is the default for ADO.
Remote Provider	This argument specifies the name of a provider to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
Remote Server	This argument specifies the path name of a server to use when opening a client-side connection (for a Remote Data Service only). This argument is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
User ID	This argument specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used by Host Integration Server to validate that the user can log on to the target host system and has appropriate access rights to the file.

The OLE DB Provider for AS/400 and VSAM also supports a number of provider-specific arguments, some of which have default values as specified in the table below. These arguments are as listed in the following table.

Argument	**Description**
BinAsChar	This parameter indicates whether to process binary fields as character fields (default is 0; do not process binary fields as character fields).
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).
Default Library	The default AS/400 library to be accessed. This parameter is not required for mainframe access and is optional when connecting to AS/400 files.
HCDFileName	The fully qualified file name of the DDM host column description (HCD) file. This parameter can be an UNC string up to 256 characters in length. A path does not need to be included in the name if the HCD file is located in the SNA system directory. This parameter is required when connecting to mainframe systems and is optional when connecting to OS/400.
Local LU	The name of the local LU alias configured in the SNA server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and SNA server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.

NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the computer. If this argument is omitted, the default value is set to Latin 1 (1252).
RDB	The remote database name for OS/400. You only need to specify this value if it is different from the remote LU alias configured in the SNA server.
Repair Host Keys	This parameter indicates whether the OLE DB provider should repair any host key values set in the registry and defaults to false.
RemoteLU	The name of the remote LU alias configured in the SNA server.
StrictVal	This parameter indicates whether strict validation should be used and defaults to false.

Note
Not all of these parameters are required. The user can also be prompted for this information.

The OLE DB Provider for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by OLE DB Provider for DB2 supplied with Host Integration Server differ from the arguments supported by the earlier OLE DB Provider for DB2 included with SNA Server 4.0.

The arguments supported by the OLE DB Provider for DB2 supplied with Host Integration Server are listed in the following table.

Argument	**Description**
BinAsChar	When this parameter is set to true, the OLE DB Provider for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The Host CCSID and PC Code Page values are required input and output parameters. This parameter defaults to false.
CCSID	The Code Character Set Identifier (CCSID) attribute indicates the character set used on the host. If this argument is omitted, the default value is U.S./Canada (37).

DefSch	The name of the default schema (collection/owner) where the system catalogs resides. This parameter can be QSYS2;SYSIBM;SYSTEM; CURLIB; or USERID depending on platform. This parameter does not have a default value.
InitCat	This parameter is used as the first part of a 3-part fully qualified table name. In DB2 (MVS, OS/390), this property is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. In DB2/400, this parameter is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then one can be created using the Add option. In DB2 Universal Database, this property is referred to as DATABASE. This parameter has no default value.
LocalLU	The name of the local LU alias configured in Host Integration Server.
ModeName	The APPC mode (must be set to a value that matches the host configuration and Host Integration Server configuration). Legal values for the APPC mode include QPCSUPP (5250), #NTER (interactive), #NTERSC (interactive), #BATCH (batch), #BATCHSC (batch), and custom modes.
NetAddr	When TCP/IP has been selected for the Network Transport Library, this parameter indicates the IP address of the host.
NetPort	When TCP/IP has been selected for the Network Transport Library, this parameter is the TCP/IP port used for communication with the source. The default value is TCP/IP port 446.
NetLib	This parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA.
PCCodePage	The character code page to use on the PC. If this argument is omitted, the default value is set to Latin 1 (1252).
PkgCol	The name of the DRDA target collection (AS/400 library) where the Microsoft OLE DB Provider for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft OLE DB Provider for DB2 uses packages to issue dynamic and static SQL statements. The OLE DB Provider will create packages dynamically in the location to which the user points using the Package Collection parameter.

RemoteLU	The name of the remote LU alias configured in Host Integration Server.
TPName	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPName is set to 0X07F9F9F9.
UOW	This parameter determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information.

The ODBC Driver for DB2 also supports a number of provider-specific arguments, some of which have default values as specified in the tables below. The arguments supported by ODBC Driver for DB2 supplied with Host Integration Server differ from the arguments supported by the earlier ODBC Driver for DB2 included with SNA Server 4.0.

The arguments supported by the ODBC Driver for DB2 supplied with Host Integration Server are listed in the following table.

Argument	**Description**
BAC	When the *BinAsChar* parameter is set to **true** (1), the ODBC Driver for DB2 treats binary data type fields (with a CCSID of 65535) as character data type fields on a per-data source basis. The **CCSID** and **PCCodePage** values are required input parameters.
CCSID	The character code set identifier (CCSID) matching the DB2 data as represented on the remote computer. The CCSID property is required when processing binary data as character data. Unless the **BinAsChar** value is set, character data is converted based on the DB2 column CCSID and default ANSI code page. If this argument is omitted, this parameter defaults to U.S./Canada (37).
CP	The character code page to use on the PC. This parameter is required when processing binary data as character data. Unless the Binary as Character (BAC) value is set, character data is converted based on the default ANSI code page configured in Windows. If this argument is omitted, the default value is set to Latin 1 (1252).

DESC	A field to provide a comment describing this ODBC data source. The description is an optional parameter and may be left blank.
DS	The Default Schema parameter is the name of the Collection where the ODBC Driver for DB2 looks for catalog information. The Default Schema is the "SCHEMA" name for the target collection of tables and views. The ODBC driver uses Default Schema to restrict results sets for popular operations, such as enumerating a list of tables in a target collection (for example, ODBC Catalog SQLTables). For DB2, the Default Schema is the target AUTHENTICATION (User ID or "owner"). For DB2/400, the Default Schema is the target COLLECTION name. For DB2 Universal Database (UDB), the Default Schema is the SCHEMA name. If the user does not provide a value for Default Schema, then the ODBC driver uses the USER_ID provided at logon. For DB2/400, the driver will use QSYS2 if there is no collection found matching the USER_ID value. Obviously, this default is inappropriate in many cases, therefore it is essential that the Default Schema value in the data source be defined.
DSN	The data source name is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file, which is stored in the Program Files\Common Files\ODBC\Data Sources directory.
LLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
MN	When SNA is used for the Network Transport Library (NTL), the Mode Name field is the APPC mode and must be set to a value that matches the host configuration and Host Integration Server configuration. Legal values for the APPC mode include QPCSUPP (common system default often used by 5250), #INTER (interactive), #INTERSC (interactive with minimal routing security), #BATCH (batch), #BATCHSC (batch with minimal routing security), #IBMRDB (DB2 remote database access), and custom modes. The following modes that support bidirectional LZ89 compression are also legal: #INTERC (interactive with compression), INTERCS (interactive with compression and minimal routing security), BATCHC (batch with compression), and BATCHCS (batch with compression and minimal routing security). This parameter normally defaults to QPCSUPP.
NA	When TCP/IP is used for the Network Transport Library (NTL), the Network Address parameter indicates the IP address or the hostname alias of the host DB2 server.
NP	When TCP/IP is used for the Network Transport Library (NTL), the Network Port parameter indicates the TCP/IP port used for communication with the target DB2 DRDA service. The default value is TCP/IP port 446.
NTL	The Network Transport Library parameter determines whether TCP/IP or SNA APPC is used for network communication. The possible values for this parameter are TCPIP or SNA. This value defaults to SNA. If the default SNA is selected, then values for LLU, MN, and RLU are required. If TCP/IP is selected, then values for NetAddr and NetPort are required.
PC	The name of the DRDA target collection (AS/400 library) where the Microsoft ODBC Driver for DB2 should store and bind DB2 packages. This could be same as the Default Schema. The Microsoft ODBC Driver for DB2, which is implemented as an IBM DRDA application requester, uses packages to issue dynamic and static SQL statements. The ODBC driver will create packages dynamically in the location to which the user points using the Package Collection parameter.
PDS	The Provider Data Source is a required parameter that is used to define the data source. The ODBC driver manager uses this attribute value to load the correct ODBC data source configuration from the registry or from a file. For File data sources, this field is used to name the DSN file, which is stored in the Program Files\Common Files\ODBC\Data Sources directory.

PROV	Specifies the name of the provider to use for the connection. To use the ODBC Driver for DB2, the Provider string must be set to "MSDASQL" or not used as part of the **ConnectionString** since this value is the default for ADO.
PWD	Specifies a valid mainframe or AS/400 password to use when opening the connection. This password is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. Note that this parameter is the same as the Parameter parameter.
RDB	The remote database name parameter is used as the first part of a three-part, fully qualified DB2 table name. This parameter is referred to by different names depending on the DB2 platform. In DB2 on MVS and OS/390, this parameter is referred to as LOCATION. The SYSIBM.LOCATIONS table lists all the accessible locations. To find the location of the DB2 to which you need to connect on these platforms, ask the administrator to look in the TSO Clist DSNTINST under the DDF definitions. These definitions are provided in the DSNTIPR panel in the DB2 Installation manual. In DB2/400 on OS/400, this property is referred to as RDBNAM. The RDBNAM value can be determined by invoking the WRKRDBDIRE command from the console to the OS/400 system. If there is no RDBNAM value, then a value can be created using the **Add** option. In DB2 Universal Database, this property is referred to as DATABASE.
RLU	When SNA is used for the network transport, this field is the name of the remote LU alias configured in Host Integration Server.
TPN	The Transaction Program (TP) Name parameter represents the default transaction program name for the DB2 DRDA application server (AS) which is 07F6DB (DB2DRDA). However, some DB2 installations may be configured to use an alternate TP name. Host Integration Server uses the alternate TP name in the off-line demo configuration (DRDADEMO.UDL). In that case, TPN is set to 0X07F9F9F9.
UID	Specifies a valid mainframe or AS/400 user name to use when opening the connection. This user name is used to validate that the user can log on to the target DB2 host system and has appropriate access rights to the database. This parameter is the same as the User ID parameter.
UOW	Determines whether two-phase commit is enabled. The possible values for this parameter are DUW (distributed unit of work) or RUW (remote unit of work). This value defaults to RUW. When this parameter is set to RUW, two-phase commit is disabled. When this parameter is set to DUW, two-phase commit is enabled in the OLE DB Provider for DB2. Distributed transactions are handled using Microsoft Transaction Server, Microsoft Distributed Transaction Coordinator, and the SNA LU 6.2 Resync Service. This option works only with DB2 for OS/390 v5R1 or later. This option also requires that SNA (LU 6.2) service is selected as the network transport and Microsoft Transaction Server (MTS) is installed.

Note

Not all of these parameters are required. The user can also be prompted for this information. **Values used for the CursorType parameter**

Thisparameter can be one of the enumerated values for **CursorTypeEnum** as listed in the following table.

Enumeration	Value	Description

adOpenUnspecified	-1	This indicates an unspecified value for the *CursorType*. This value is not supported by the Microsoft OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adOpenForwardOnly	0	Specifying this value opens a forward-only-type cursor. This *CursorType* is identical to a static cursor, except that you can only scroll forward through records. This improves performance when only one pass through a **Recordset** is needed. This value is not supported by the Microsoft OLE DB Provider for AS/400 and VSAM.
adOpenKeyset	1	Specifying this value opens a keyset-type cursor. This *CursorType is* similar to a dynamic cursor with a few exceptions. Records that other users delete are inaccessible from your **Recordset**. Data changes to existing records by other users are still visible, but records added by other users are not visible (cannot be seen). This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adOpenDynamic	2	Specifying this value opens a dynamic-type cursor. Additions, changes, and deletions by other users are visible, and all types of movement through the recordset are allowed, except for bookmarks if the provider does not support them. A dynamic cursor is the only *CursorType* supported by the OLE DB Provider for AS/400 and VSAM.
adOpenStatic	3	Specifying this value opens a static-type cursor. A static cursor provides a static copy of a set of records that can be used to find data or generate reports. Additions, changes, or deletions by other users are not visible with a static cursor. This value is not supported by the OLE DB Provider for AS/400 and VSAM.

This optional argument defaults to **adOpenForwardOnly**, a value that is mapped to **adOpenDynamic** by the OLE DB provider for AS/400 and VSAM.

Values used for the LockType parameter

This parameter can be one of the enumerated values for **LockTypeEnum** as listed in the following table.

Enumeration	Value	Description
adLockUnspecified	-1	Indicates an unspecified value for the *LockType*. This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adLockReadOnly	1	Specifying this value opens a **Recordset** object read-only and data cannot be altered.
adLockPessimistic	2	Specifying this value opens a recordset with pessimistic locking. Record-by-record, the OLE DB Provider does whatever is necessary to ensure successful editing of the records, usually by locking records at the data source immediately upon editing. This lock type is supported by the OLE DB Provider for AS/400 and VSAM and the OLE DB Provider. However, the OLE DB Provider for AS/400 and VSAM internally maps this lock type to **adLockBatchOptimistic**.
adLockOptimistic	3	Specifying this value opens a recordset with optimistic locking. Record-by-record, the OLE DB Provider locks records only when the **Update** method is invoked on a **Recordset** object. This lock type is not supported by the OLE DB Provider for DB2.

adLockBatchOptimistic	4	Specifying this value opens a **Recordset** with batch optimistic locking. This value is required for batch update mode as opposed to immediate update. This lock type is not supported by the OLE DB Provider for DB2.

This optional argument defaults to **adLockReadOnly**.

Values used by the Optionsproperty

The **CommandTypeEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adCmdUnspecified	-1	This value indicates that the **CommandText** property is unspecified. This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adCmdText	1	This value evaluates the **CommandText** property as a textual definition of a command or stored procedure call.
adCmdTable	2	This value evaluates the **CommandText** property as a table name. This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adCmdStoredProc	4	This value evaluates the **CommandText** property as a stored procedure name. This value is not supported by the OLE DB Provider for AS/400 and VSAM.
adCmdUnknown	8	This value indicates that the type of command in **CommandText** property is not known. This is the default value. This value is not supported by the OLE DB Provider for AS/400 and VSAM.

Remarks

The **Open** method on a **Recordset** object is used to open tables on a remote DDM server. Using the **Open** method on a **Recordset** object establishes the physical connection to a data source and opens a cursor that represents records from a base table or the results of a query. After this method successfully completes, the **Recordset** object is live and other methods can be invoked on the **Recordset** object to process results.

The optional *Source* parameter is used to specify the command text required to open a data file on the host using the OLE DB Provider for AS/400 and VSAM. The syntax in this case is as follows:

```
recordset = CreateObject("ADODB.Recordset.2.0")
recordset.Open "EXEC OPEN LIBRARY/FILE", connection, adOpenDynamic, adLockOptimistic, adCmd
Text
```

Using the OLE DB Provider for AS/400 and VSAM, the *Source* parameter represents a table name and uses one of the following host file naming conventions.

Host file type	File naming convention
VSAM Data Sets	DATASETNAME.FILENAME
Partitioned Data Sets	DATASETNAME.FILENAME(MEMBER)
OS/400 Files	LIBRARY/FILE
OS/400 Files	LIBRARY/FILE.NAME
OS/400 File Members	LIBRARY/FILE(MEMBER)

OS/400 File Members	LIBRARY/FILE.NAME(MEMBER)

Note that if a member of a library contains a dot in the member name, the member name must be surrounded by double quotes. For example, if the member name is NAMES.DAT, the proper syntax for command text used for the **Recordset.Open** method is:

```
recordset = CreateObject("ADODB.Recordset.2.0")
recordset.Open "EXEC OPEN LIBRARY/FILE(""NAMES.DAT"")", connection, adOpenDynamic, adLockOp
timistic, adCmdText
```

Note the doubled quotes are required surrounding the member name in this example since the member name contains a period. The full path to the mainframe data set must be specified when using the OLE DB Provider for AS/400 and VSAM. In the example above, there are two path elements (LIBRARY/FILE) and one name element (NAMES.DAT).

Whenever a VSAM data set is allocated, it is given a unique name composed of one or more segments. Each segment of the data set name is joined by periods and represents a level of qualification. For example, the following data set has four segments that comprise the fully-qualified data set name (three path elements and one name element):

```
SAMPLES.DEMO.KSDS.TITLES
```

The high-level qualifier is SAMPLES. The low-level qualifier is TITLES. Each segment can be from 1-8 characters in length (the first character must be alphabetic, while the remainder can be alphanumeric or hyphens). The full data set name must be no more than 44 characters in length and contain no more than 22 segments.

The optional *ActiveConnection* parameter corresponds to the **ActiveConnection** property on a **Recordset** object and specifies on which connection to open the **Recordset** object. If a connection string definition is passed for this argument, ADO opens a new connection using the specified parameters. The value of this **ActiveConnection** property can be changed after opening the **Recordset** to send updates to another provider. The **ActiveConnection** property is set to **Nothing** (in Microsoft® Visual Basic®) to disconnect the recordset from the OLE DB Provider. If the optional *ActiveConnection* parameter is used to specify a connection string, this string must contain a series of *argument=value* statements separated by semicolons.

The **ActiveConnection** property on a **Recordset** object automatically inherits the value used for the *ActiveConnection* parameter. Therefore, the **ActiveConnection** property of the **Recordset** object can be set before opening the **Recordset** object, or the *ActiveConnection* parameter can be used to set or override the current connection parameters during the **Open** method call.

The *CursorType* parameter cannot be omitted using the OLE DB Provider for AS/400 and VSAM since this parameter defaults to **adOpenForwardOnly**, a *CursorType* that is not supported on the OLE DB Provider. The *CursorType* parameter must be set to **adOpenDynamic**, otherwise an error will occur and results will be unpredictable.

There are a number of different ways to open a recordset and connect to a data source. The **Open** method of the **Recordset** object can pass all of the appropriate connection information as part of the *ActiveConnection* parameter or by setting the **ActiveConnection** property of the **Recordset** object, if this information is known in advance. The syntax, in this case using the *ActiveConnection* parameter and the OLE DB Provider for AS/400 and VSAM, is as follows:

```
recordset = CreateObject("ADODB.Recordset.2.0")
recordset.Open "EXEC OPEN LIBRARY/FILE","Provider=SNAOLEDB;Data Source=REMLU;User ID=USERNA
ME;Password=password;Local LU=LOCAL;Remote LU=DATABASE;ModeName=QPCSUPP;CCSID=37;CodePage=4
37", adOpenDynamic, adLockOptimistic, adCmdText
```

Note

Not all of these parameters are required. The registry settings for the Data Source usually have default values set for remote LU, local LU, APPC mode, CCSID, and CodePage. If a data source is specified, this other information is not usually needed. The se registry settings are configured by using the Microsoft Management Console snap-in for the OLE DB Provider for AS/400 and VSAM.

For the other parameters that correspond directly to the properties of a **Recordset** object (*Source*, *CursorType*, and *LockType*), the relationships of the parameters to the properties are:

- The property is read/write before the **Recordset** object is opened.
- The property settings are used unless the corresponding parameters are passed when executing the **Open** method. If a parameter is passed, it overrides the corresponding property setting, and the property setting is updated with the parameter value.
- After the **Recordset** object is opened, these properties become read-only.

Note
For **Recordset** objects whose **Source** property is set to a valid **Command** object, the **ActiveConnection** property is read-only, even if the **Recordset** object is not open.

If a **Command** object is passed in the *Source* parameter and an *ActiveConnection* parameter is also passed, an error occurs. The **ActiveConnection** property of the **Command** object must already be set to a valid **Connection** object or connection string.

If a **Command** object is not passed in the *Source* argument, the *Options* argument must be set to **adCmdText**. If the *Options* argument is not defined, you may experience diminished performance because ADO must make calls to the OLE DB Provider to determine if the argument is a command statement. If you know what *Source* type you are using, setting the *Options* argument instructs ADO to jump directly to the relevant code.

If the data source returns no records, the provider sets both the **BOF** and **EOF** properties on the **Recordset** object to **True**, and the current record position is undefined. You can still add new data to this empty **Recordset** object if the cursor type allows it.

When operations have been concluded over an open **Recordset** object, the **Close** method should be invoked on the **Recordset** object to free any associated system resources. Closing a **Recordset** object does not remove it from memory; you may change its property settings and use the **Open** method to open it again later. To completely eliminate an object from memory, set the **Recordset** object variable to **Nothing**.

If errors occur, these can be examined with the **Errors** collection on the **Recordset** object.

OpenSchema Method (ADO)

The **OpenSchema** method on a **Connection** object obtains database schema information from the provider.

Syntax

```
recordset = connection.OpenSchema ( QueryType, Criteria, SchemaID )
```

Parameters

QueryType

This parameter specifies a **SchemaEnum** value that indicates the type of schema query to run.

The **SchemaEnum** values supported by the Microsoft® OLE DB Provider for AS/400 and VSAM can be one of the constants listed in the table following the Parameters section.

Criteria

This optional parameter specifies an array of query constraints for each *QueryType* option, as listed below.

The values supported by the OLE DB Provider for AS/400 and VSAM can be one of the constants listed in the table following the Parameters section. This value depends on the *QueryType*.

Note

The **adSchemaindexes** TYPE restriction is not supported by the OLE DB Provider for DB2.

Note

The **adSchemaProcedures** PROCEDURE_SCHEMA, and **adSchemaProcedureParameters** PROCEDURE_SCHEMA restrictions are not supported when connecting to DB/2 on OS/390 platforms.

SchemaID

This optional parameter specifies the GUID for a provider-schema schema query not defined by the OLE DB specification. This parameter is required if the *QueryType* parameter is set to **adSchemaProviderSpecific**; otherwise, it is not used. This parameter is not supported by the OLE DB Provider for AS/400 and VSAM.

Values for QueryType

Enumeration	Value	Description
adSchemaColumns	4	This value indicates that the *QueryType* is requesting column information for tables on the server (not supported when connecting to mainframes).
adSchemaIndexes	12	This value indicates that the *QueryType* is requesting index information about the tables on the server (not supported when connecting to mainframes).
adSchemaTables	20	This value indicates that the *QueryType* is requesting information about the tables on the server.
adSchemaProviderTypes	22	This value indicates that the *QueryType* is requesting provider-type information.

The **SchemaEnum** values supported by the Microsoft OLE DB Provider for DB2 and the Microsoft ODBC Driver for DB2 can be one of the constants listed in the following table.

Enumeration	Value	Description
adSchemaColumns	4	This value indicates that the *QueryType* is requesting column information for tables on the server (not supported when connecting to mainframes).

adSchemaIndexes	12	This value indicates that the *QueryType* is requesting index information about the tables on the server (not supported when connecting to mainframes).
adSchemaProcedures	16	This value indicates that the *QueryType* is requesting information about stored procedures on the server.
adSchemaTables	20	This value indicates that the *QueryType* is requesting information about the tables on the server.
adSchemaProviderTypes	22	This value indicates that the *QueryType* is requesting provider-type information.
adSchemaProcedureParameters	26	This value indicates that the *QueryType* is requesting information about parameters used by stored procedures on the server.
adSchemaPrimaryKeys	28	This value indicates that the *QueryType* is requesting information about the primary keys for tables on the server.

Values for Criteria

QueryType / Enumeration
adSchemaColumns
TABLE_CATALOG
TABLE_SCHEMA
TABLE_NAME
COLUMN_NAME
adSchemaIndexes
TABLE_CATALOG
TABLE_SCHEMA
INDEX_NAME
TYPE
TABLE_NAME
adSchemaTables
TABLE_CATALOG
TABLE_SCHEMA
TABLE_NAME
TABLE_TYPE
adSchemaProviderTypes
DATA_TYPE

BEST_MATCH

The values supported by the OLE DB Provider for DB2 and the ODBC Driver for DB2 can be one of the constants listed in the following table, depending on the *QueryType*.

QueryType / Enumeration
adSchemaColumns
TABLE_CATALOG
TABLE_SCHEMA
TABLE_NAME
COLUMN_NAME
adSchemaIndexes
TABLE_CATALOG
TABLE_SCHEMA
INDEX_NAME
TABLE_NAME
adSchemaPrimaryKeys
TABLE_CATALOG
TABLE_SCHEMA
TABLE_NAME
adSchemaProcedures
PROCEDURE_CATALOG
PROCEDURE_SCHEMA (see Notes)
PROCEDURE_NAME
PROCEDURE_TYPE
adSchemaProcedureParameters
PROCEDURE_CATALOG
PROCEDURE_SCHEMA (see Notes)
PROCEDURE_NAME
PROCEDURE_TYPE

adSchemaProviderTypes
DATA_TYPE
BEST_MATCH
adSchemaTables
TABLE_CATALOG
TABLE_SCHEMA
TABLE_NAME
TABLE_TYPE

Return Value

Returns a **Recordset** object that contains schema information requested.

Remarks

The **OpenSchema** method on a **Connection** object is used to return information about the data source, such as information about the tables on the server and the columns in the tables.

The *Criteria* argument is an array of values that can be used to limit the results of a schema query. Each schema query supports a different set of parameters. The actual schemas are defined by the OLE DB specification under the **IDBSchemaRowset** interface. The schema queries supported in ActiveX® Data Objects (ADO) version 1.5 and later by the OLE DB Provider for AS/400 and VSAM are listed above.

The **OpenSchema** method allows an application to pass at run time the target library of a Partitioned Data Set (PDS/PDSE), a dataset, or a member name as one of the *Criteria* array arguments to retrieve the schema.

Providers are not required to support all of the OLE DB standard schema *QueryType* values. Specifically, only **adSchemaTables**, **adSchemaColumns**, and **adSchemaProviderTypes** are required by the OLE DB specification. However, the provider is not required to support the *Criteria* constraints listed above for those schema queries. Support for other schema *QueryType* values is optional.

The schema information specified in OLE DB is based on the assumption that providers support the concepts of a catalog and a schema. The ANSI SQL 92 specification defines them as follows:

- A catalog contains one or more schemas, but always contains a schema named INFORMATION_SCHEMA which contains the views and domains of the information schema. In Microsoft SQL Server™ and Microsoft Access terms, a catalog is a database; in ODBC 2.*x* terms, a catalog is a qualifier.

- A schema is a collection of database objects that are owned or have been created by a particular user. In Microsoft SQL Server and ODBC 2.*x* terms, a schema is an owner; there is no equivalent to a schema in a Microsoft Access database.

Schema information in ADO and OLE DB is retrieved using predefined schema rowsets.

In This Section

- adSchemaColumns
- adSchemaIndexes
- adSchemaPrimaryKeys
- adSchemaProcedures
- adSchemaProcedureParameters

- adSchemaProviderTypes
- adSchemaTables

adSchemaColumns

The **adSchemaColumns** *QueryType* identifies the columns of tables defined in the catalog that are accessible to a given user. This *QueryType* is supported by the Microsoft OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaColumns** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
TABLE_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
TABLE_SCHEMA	DBTYPE_WSTR	Unqualified schema name. NULL if the provider does not support schemas.
TABLE_NAME	DBTYPE_WSTR	Table name.
COLUMN_NAME	DBTYPE_WSTR	The name of the column; this might not be unique. If this cannot be determined, a NULL is returned. This column, together with the COLUMN_GUID and COLUMN_PROPID columns, forms the column ID. One or more of these columns will be NULL depending on which elements of the DBID structure the provider uses. If possible, the resulting column ID should be persistent. However, some providers do not support persistent identifiers for columns. The column ID of a base table should be invariant under views.
COLUMN_GUID	DBTYPE_GUID	Column GUID.
COLUMN_PROPID	DBTYPE_UI4	Column property ID.

ORDINAL_POSITION	DBTYPE_UI4	The ordinal of the column. Columns are numbered starting from one. NULL if there is no stable ordinal value for the column.
COLUMN_HASDEFAULT	DBTYPE_BOOL	VARIANT_TRUE: The column has a default value. VARIANT_FALSE: The column does not have a default value or it is unknown whether the column has a default value.
COLUMN_DEFAULT	DBTYPE_WSTR	Default value of the column. A provider may expose DBCOLUMN_DEFAULTVALUE but not DBCOLUMN_HASDEFAULT (for SQL 92 tables) in the rowset returned by **IColumnsRowset::GetColumnsRowset**. If the default value is the NULL value, COLUMN_HASDEFAULT is VARIANT_TRUE, and the COLUMN_DEFAULT column is a NULL value.
COLUMN_FLAGS	DBTYPE_UI4	A bitmask that describes column characteristics. The DBCOLUMNFLAGS enumerated type specifies the bits in the bitmask. For information about DBCOLUMNFLAGS, see **IColumnsInfo::GetColumnInfo**. This column cannot contain a NULL value.
IS_NULLABLE	DBTYPE_BOOL	VARIANT_TRUE: The column might be nullable. VARIANT_FALSE: The column is known not to be nullable.
DATA_TYPE	DBTYPE_UI2	The indicator of the columns data type. If the data type of the column varies from row to row, this must be DBTYPE_VARIANT.
TYPE_GUID	DBTYPE_GUID	The GUID of the columns data type.
CHARACTER_MAXIMUM_LENGTH	DBTYPE_UI4	The maximum possible length of a value in the column. For character, binary, or bit columns, this is one of the following: The maximum length of the column in characters, bytes, or bits, respectively, if one is defined. For example, a CHAR(5) column in an SQL table has a maximum length of five (5). The maximum length of the data type in characters, bytes, or bits, respectively, if the column does not have a defined length. Zero (0) if neither the column nor the data type has a defined maximum length. NULL for all other types of columns.

CHARACTER_OCTET_LENGTH	DBTYPE_UI4	Maximum length in octets (bytes) of the column, if the type of the column is character or binary. A value of zero means the column has no maximum length. NULL for all other types of columns.
NUMERIC_PRECISION	DBTYPE_UI2	If the columns data type is numeric, this is the maximum precision of the column. The precision of columns with a data type of DBTYPE_DECIMAL or DBTYPE_NUMERIC depends on the definition of the column. If the columns data type is not numeric, this is NULL.
NUMERIC_SCALE	DBTYPE_I2	If columns type indicator is DBTYPE_DECIMAL or DBTYPE_NUMERIC, this is the number of digits to the right of the decimal point. Otherwise, this is NULL.
DATETIME_PRECISION	DBTYPE_UI4	Datetime precision (number of digits in the fractional seconds portion) of the column if the column is a datetime or interval type.
CHARACTER_SET_CATALOG	DBTYPE_WSTR	Catalog name in which the character set is defined. NULL if the provider does not support catalogs or different character sets.
CHARACTER_SET_SCHEMA	DBTYPE_WSTR	Unqualified schema name in which the character set is defined. NULL if the provider does not support schemas or different character sets.
CHARACTER_SET_NAME	DBTYPE_WSTR	Character set name. NULL if the provider does not support different character sets.
COLLATION_CATALOG	DBTYPE_WSTR	Catalog name in which the collation is defined. NULL if the provider does not support catalogs or different collations.
COLLATION_SCHEMA	DBTYPE_WSTR	Unqualified schema name in which the collation is defined. NULL if the provider does not support schemas or different collations.

COLLATION_NAME	DBTYPE_WSTR	Collation name. NULL if the provider does not support different collations.
DOMAIN_CATALOG	DBTYPE_WSTR	Catalog name in which the domain is defined. NULL if the provider does not support catalogs or domains.
DOMAIN_SCHEMA	DBTYPE_WSTR	Unqualified schema name in which the domain is defined. NULL if the provider does not support schemas or domains.
DOMAIN_NAME	DBTYPE_WSTR	Domain name. NULL if the provider does not support domains.
DESCRIPTION	DBTYPE_WSTR	Human-readable description of the column. For example, the description for a column named Name in the Employee table might be "Employee name."

The default sort order for the **adSchemaColumns** rowset is TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.

adSchemaIndexes

The **adSchemaIndexes** *QueryType* identifies the indexes defined in the catalog that are owned by a given user. This *QueryType* is supported by the Microsoft OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaIndexes** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
TABLE_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
TABLE_SCHEMA	DBTYPE_WSTR	Unqualified schema name. NULL if the provider does not support schemas.
TABLE_NAME	DBTYPE_WSTR	Table name.

INDEX_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
INDEX_SCHEMA	DBTYPE_WSTR	Unqualified schema name. NULL if the provider does not support schemas.
INDEX_NAME	DBTYPE_WSTR	Index name.
PRIMARY_KEY	DBTYPE_BOOL	Whether the index represents the primary key on the table. NULL if this is not known.
UNIQUE	DBTYPE_BOOL	Whether index keys must be unique. One of the following: VARIANT_TRUE: The index keys must be unique. VARIANT_FALSE: Duplicate keys are allowed.

CLUSTERED	DBTYPE_BOOL	Whether an index is clustered. One of the following: VARIANT_TRUE: The leaf nodes of the index contain full rows, not bookmarks. This is a way to represent a table clustered by key value. VARIANT_FALSE: The leaf nodes of the index contain bookmarks of the base table rows whose key value matches the key value of the index entry.
TYPE	DBTYPE_UI2	The type of the index. One of the following: DBPROPVAL_IT_BTREE: The index is a B+-tree. DBPROPVAL_IT_HASH: The index is a hash file using, for example, linear or extensible hashing. DBPROPVAL_IT_CONTENT: The index is a content index. DBPROPVAL_IT_OTHER: The index is some other type of index.
FILL_FACTOR	DBTYPE_I4	For a B+-tree index, this property represents the storage utilization factor of page nodes during the creation of the index. The value is an integer from 1 to 100 representing the percentage of use of an index node. For a linear hash index, this property represents the storage utilization of the entire hash structure (the ratio of used area to total allocated area) before a file structure expansion occurs.
INITIAL_SIZE	DBTYPE_I4	The total amount of bytes allocated to this structure at creation time.
NULLS	DBTYPE_I4	Whether null keys are allowed. One of the following: DBPROPVAL_IN_DISALLOWNULL: The index does not allow entries where the key columns are NULL. If the consumer attempts to insert an index entry with a NULL key, then the provider returns an error. DBPROPVAL_IN_IGNORENULL: The index does not insert entries containing NULL keys. If the consumer attempts to insert an index entry with a NULL key, then the provider ignores that entry and no error code is returned. DBPROPVAL_IN_IGNOREANYNULL: The index does not insert entries where some column key has a NULL value. For an index having a multicolumn search key, if the consumer inserts an index entry with NULL value in some column of the search key, then the provider ignores that entry and no error code is returned.
SORT_BOOKMARKS	DBTYPE_BOOL	How the index treats repeated keys. One of the following: VARIANT_TRUE: The index sorts repeated keys by bookmark. VARIANT_FALSE: The index does not sort repeated keys by bookmark.

AUTO_UPDATE	DBTYPE_BOOL	Whether the index is maintained automatically when changes are made to the corresponding base table. One of the following: VARIANT_TRUE: The index is automatically maintained. VARIANT_FALSE: The index must be maintained by the consumer through explicit calls to **IRowsetChange**. Ensuring consistency of the index as a result of updates to the associated base table is the responsibility of the consumer.
NULL_COLLATION	DBTYPE_I4	How NULLs are collated in the index. One of the following: DBPROPVAL_NC_END: NULLs are collated at the end of the list, regardless of the collation order. DBPROPVAL_NC_START: NULLs are collated at the start of the list, regardless of the collation order. DBPROPVAL_NC_HIGH: NULLs are collated at the high end of the list. DBPROPVAL_NC_LOW: NULLs are collated at the low end of the list.
ORDINAL_POSITION	DBTYPE_UI4	Ordinal position of the column in the index, starting with one.
COLUMN_NAME	DBTYPE_WSTR	Column name. This column, together with the COLUMN_GUID and COLUMN_PROPID columns, forms the column ID. One or more of these columns will be NULL depending on which elements of the DBID structure the provider uses.
COLUMN_GUID	DBTYPE_GUID	Column GUID.

COLUMN_PROPID	DBTYPE_UI4	Column property ID.
COLLATION	DBTYPE_I2	One of the following: DB_COLLATION_ASC: The sort sequence for the column is ascending. DB_COLLATION_DESC: The sort sequence for the column is descending. NULL: A column sort sequence is not supported.
CARDINALITY	DBTYPE_I4	Number of unique values in the index.
PAGES	DBTYPE_I4	Number of pages used to store the index.
FILTER_CONDITION	DBTYPE_WSTR	The WHERE clause identifying the filtering restriction.

The default sort order for the **adSchemaIndexes** rowset is UNIQUE, TYPE, INDEX_CATALOG, INDEX_SCHEMA, INDEX_NAME, and ORDINAL_POSITION.

adSchemaPrimaryKeys

The **adSchemaPrimaryKeys** *QueryType* identifies the primary key columns defined in the catalog by a given user. This *QueryType* is supported by the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaPrimaryKeys** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
TABLE_CATALOG	DBTYPE_WSTR	Catalog name in which the table is defined. NULL if the provider does not support catalogs.
TABLE_SCHEMA	DBTYPE_WSTR	Unqualified schema name in which the table is defined. NULL if the provider does not support schemas.
TABLE_NAME	DBTYPE_WSTR	Table name.
COLUMN_NAME	DBTYPE_WSTR	Primary key column name. This column, together with the COLUMN_GUID and COLUMN_PROPID columns, forms the column ID. One or more of these columns will be NULL depending on which elements of the DBID structure the provider uses.
COLUMN_GUID	DBTYPE_GUID	Primary key column GUID.
COLUMN_PROPID	DBTYPE_UI4	Primary key column property ID.
ORDINAL	DBTYPE_UI4	The order of the column names (and GUIDs and property IDs) in the key.
PK_NAME	DBTYPE_WSTR	Primary key name. NULL if the provider does not support primary key constraints.

The default sort order for the **adSchemaPrimaryKeys** rowset is UNIQUE, TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.

adSchemaProcedures

The **adSchemaProcedures** *QueryType* identifies information about the columns of rowsets returned by procedures. This *QueryType* is supported by the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaProcedures** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
PROCEDURE_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
PROCEDURE_SCHEMA	DBTYPE_WSTR	Unqualified schema name. NULL if the provider does not support schemas.
PROCEDURE_NAME	DBTYPE_WSTR	Table name.
COLUMN_NAME	DBTYPE_WSTR	The name of the column; this might not be unique. If this cannot be determined, a NULL is returned. This column, together with the COLUMN_GUID and COLUMN_PROPID columns, forms the column ID. One or more of these columns will be NULL depending on which elements of the DBID structure the provider uses. If possible, the resulting column ID should be persistent. However, some providers do not support persistent identifiers for columns. The column ID of a base table should be invariant under views.
COLUMN_GUID	DBTYPE_GUID	Column GUID.
COLUMN_PROPID	DBTYPE_UI4	Column property ID.

ROWSET_NUMBER	DBTYPE_UI4	Number of the rowset containing the column. This is greater than one only if the procedure returns multiple rowsets.
ORDINAL_POSITION	DBTYPE_UI4	The ordinal of the column. Columns are numbered starting from one. NULL if there is no stable ordinal value for the column.
IS_NULLABLE	DBTYPE_BOOL	VARIANT_TRUE: The column might be nullable. VARIANT_FALSE: The column is known not to be nullable.
DATA_TYPE	DBTYPE_UI2	The indicator of the columns data type. If the data type of the column varies from row to row, this must be DBTYPE_VARIANT.
TYPE_GUID	DBTYPE_GUID	The GUID of the columns data type.
CHARACTER_MAXIMUM_LENGTH	DBTYPE_UI4	The maximum possible length of a value in the column. For character, binary, or bit columns, this is one of the following: The maximum length of the column in characters, bytes, or bits, respectively, if one is defined. For example, a CHAR(5) column in an SQL table has a maximum length of five (5). The maximum length of the data type in characters, bytes, or bits, respectively, if the column does not have a defined length. Zero (0) if neither the column nor the data type has a defined maximum length. NULL for all other types of columns.
CHARACTER_OCTET_LENGTH	DBTYPE_UI4	Maximum length in octets (bytes) of the column, if the type of the column is character or binary. A value of zero means the column has no maximum length. NULL for all other types of columns.
NUMERIC_PRECISION	DBTYPE_UI2	If the columns data type is numeric, this is the maximum precision of the column. The precision of columns with a data type of DBTYPE_DECIMAL or DBTYPE_NUMERIC depends on the definition of the column. If the columns data type is not numeric, this is NULL.

NUMERIC_SCALE	DBTYPE_I2	If columns type indicator is DBTYPE_DECIMAL or DBTYPE_NUMERIC, this is the number of digits to the right of the decimal point. Otherwise, this is NULL.
DESCRIPTION	DBTYPE_WSTR	Human-readable description of the column. For example, the description for a column named Name in the Employee table might be "Employee name."

The default sort order for the **adSchemaProcedures** rowset is PROCEDURE_CATALOG, PROCEDURE_SCHEMA, and PROCEDURE_NAME.

adSchemaProcedureParameters

The **adSchemaProcedureParameters** *QueryType* identifies information about the parameters and return codes of procedures. This *QueryType* is supported by the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaProcedureParameters** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
PROCEDURE_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
PROCEDURE_SCHEMA	DBTYPE_WSTR	Unqualified schema name. NULL if the provider does not support schemas.
PROCEDURE_NAME	DBTYPE_WSTR	Table name.
PARAMETER_NAME	DBTYPE_WSTR	Parameter name. NULL if the parameter is not named.
ORDINAL_POSITION	DBTYPE_UI2	If the parameter is an input, input/output, or output parameter, this is the one-based ordinal position of the parameter in the procedure call. If the parameter is the return value, this is zero.
PARAMETER_TYPE	DBTYPE_UI2	One of the following: DBPARAMTYPE_INPUT—The parameter is an input parameter. DBPARAMTYPE_INPUTOUTPUT—The parameter is an input/output parameter. DBPARAMTYPE_OUTPUT—The parameter is an output parameter. DBPARAMTYPE_RETURNVALUE—The parameter is a procedure return value. If the provider cannot determine the parameter type, this is NULL.
PARAMETER_HASDEFAULT	DBTYPE_BOOL	VARIANT_TRUE: The parameter has a default value. VARIANT_FALSE: The parameter does not have a default value or it is unknown whether the parameter has a default value.
PARAMETER_DEFAULT	DBTYPE_WSTR	Default value of the parameter. If the default value is the NULL value, PARAMETER_HASDEFAULT is VARIANT_TRUE, and the PARAMETER_DEFAULT value is a NULL value.
IS_NULLABLE	DBTYPE_BOOL	VARIANT_TRUE: The parameter might be nullable. VARIANT_FALSE: The parameter is not nullable.
DATA_TYPE	DBTYPE_UI2	The indicator of the parameters data type.

CHARACTER_MAXIMUM_LENGTH	DBTYPE_UI4	The maximum possible length of a value in the parameter. For character, binary, or bit columns, this is one of the following: The maximum length of the parameter in characters, bytes, or bits, respectively, if one is defined. For example, a CHAR(5) column in an SQL table has a maximum length of five (5). The maximum length of the data type in characters, bytes, or bits, respectively, if the parameter does not have a defined length. Zero (0) if neither the parameter nor the data type has a defined maximum length. NULL for all other types of parameters.
CHARACTER_OCTET_LENGTH	DBTYPE_UI4	Maximum length in octets (bytes) of the parameter, if the type of the parameter is character or binary. A value of zero means the parameter has no maximum length. NULL for all other types of parameter.
NUMERIC_PRECISION	DBTYPE_UI2	If the parameters data type is numeric, this is the maximum precision of the parameter. The precision of parameters with a data type of DBTYPE_DECIMAL or DBTYPE_NUMERIC depends on the definition of the parameters. If the parameters data type is not numeric, this is NULL.
NUMERIC_SCALE	DBTYPE_I2	If parameters type indicator is DBTYPE_DECIMAL or DBTYPE_NUMERIC, this is the number of digits to the right of the decimal point. Otherwise, this is NULL.
DESCRIPTION	DBTYPE_WSTR	Human-readable description of the parameter. For example, the description for a parameter named Name in a procedure that adds a new employee might be "Employee name."
TYPE_NAME	DBTYPE_WSTR	Provider-specific data type name.
LOCAL_TYPE_NAME	DBTYPE_WSTR	Localized version of TYPE_NAME. NULL is returned if a localized name is not supported by the data provider.

The default sort order for the **adSchemaProcedureParameters** rowset is PROCEDURE_CATALOG, PROCEDURE_SCHEMA, and PROCEDURE_NAME.

adSchemaProviderTypes

The **adSchemaProviderTypes** *QueryType* identifies the data types supported by the data provider. This *QueryType* is supported by the Microsoft OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaProviderType** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
TYPE_NAME	DBTYPE_WSTR	Provider-specific data type name.
DATA_TYPE	DBTYPE_UI2	The indicator of the data type.
COLUMN_SIZE	DBTYPE_UI4	The length of a non-numeric column or parameter refers to either the maximum or the defined length for this type by th e provider. For character data, this is the maximum or defined length in characters. If the data type is numeric, this is the upper bound on the maximum precision of the data type.
LITERAL_PREFIX	DBTYPE_WSTR	Character or characters used to prefix a literal of this type in a text command.

LITERAL_SUFFIX	DBTYPE_WSTR	Character or characters used to suffix a literal of this type in a text command.
CREATE_PARAMS	DBTYPE_WSTR	The creation parameters are specified by the consumer when creating a column of this data type. For example, the SQL data type DECIMAL needs a precision and a scale. In this case, the creation parameters might be the string "precision,scale". In a text command, to create a DECIMAL column with a precision of 10 and a scale of 2, the value of the TYPE_NAME column might be DECIMAL() and the complete type specification would be DECIMAL(10,2). The creation parameters appear as a comma-separated list of values, in the order they are to be supplied, with no surrounding parentheses. If a creation parameter is length, maximum length, precision, or scale, "length", "max length", "precision", and "scale" should be used, respectively. If the creation parameters are some other value, it is provider-specific what text is used to describe the creation parameter. If the data type requires creation parameters, "()" generally appears in the type name. This indicates the position at which to insert the creation parameters. If the type name does not include "()", the creation parameters are enclosed in parentheses and appended to the end of the data type name.
IS_NULLABLE	DBTYPE_BOOL	VARIANT_TRUE: The data type is nullable. VARIANT_FALSE: The data type is not nullable. NULL: It is not known whether the data type is nullable.
CASE_SENSITIVE	DBTYPE_BOOL	VARIANT_TRUE: The data type is a character type and is case-sensitive. VARIANT_FALSE: The data type is not a character type or is not case-sensitive.
SEARCHABLE	DBTYPE_UI4	If the provider supports **ICommandText**, then this column is an integer indicating the searchability of a data type, otherwise this column is NULL. One of the following: DB_UNSEARCHABLE: The data type cannot be used in a WHERE clause. DB_LIKE_ONLY: The data type can be used in a WHERE clause only with the LIKE predicate. DB_ALL_EXCEPT_LIKE: The data type can be used in a WHERE clause with all comparison operators except LIKE. DB_SEARCHABLE: The data type can be used in a WHERE clause with any comparison operator.

UNSIGNED_ATTRIBUTE	DBTYPE_BOOL	VARIANT_TRUE: The data type is unsigned. VARIANT_FALSE: The data type is signed. NULL: Not applicable to data type.
FIXED_PREC_SCALE	DBTYPE_BOOL	VARIANT_TRUE: The data type has a fixed precision and scale. VARIANT_FALSE: The data type does not have a fixed precision and scale.
AUTO_UNIQUE_VALUE	DBTYPE_BOOL	VARIANT_TRUE: Values of this type can be auto-incrementing. VARIANT_FALSE: Values of this type cannot be auto-incrementing.

LOCAL_TYPE_NAME	DBTYPE_WSTR	Localized version of TYPE_NAME. NULL is returned if a localized name is not supported by the data provider.
MINIMUM_SCALE	DBTYPE_I2	If the type indicator is DBTYPE_DECIMAL or DBTYPE_NUMERIC, this is the minimum number of digits allowed to the right of the decimal point. Otherwise, this is NULL.
MAXIMUM_SCALE	DBTYPE_I2	If the type indicator is DBTYPE_DECIMAL or DBTYPE_NUMERIC, this is the maximum number of digits allowed to the right of the decimal point. Otherwise, this is NULL.
GUID	DBTYPE_GUID	The GUID of the type. All types supported by a provider are described in a type library, so each type has a corresponding GUID.

TYPELIB	DBTYPE_WSTR	The type library containing the description of this type. All types supported by a provider are described in one or more type libraries.
VERSION	DBTYPE_WSTR	The version of the type definition. Providers may want to version type definitions. Different providers may use different version schemes, such as a timestamp or number (integer or float). NULL if not supported.
IS_LONG	DBTYPE_BOOL	VARIANT_TRUE: The data type is a BLOB that contains very long data; the definition of very long data is provider-specific. VARIANT_FALSE: The data type is a BLOB that does not contain very long data or is not a BLOB. This value determines the setting of the DBCOLUMNFLAGS_ISLONG flag returned by **GetColumnInfo** in **IColumnsInfo** and **GetParameterInfo** in **ICommandWithParameters**. For more information, see **GetColumnInfo** and **GetParameterInfo**.
BEST_MATCH	DBTYPE_BOOL	VARIANT_TRUE: The data type is the best match between all data types in the data source and the OLE DB data type indicated by the value in the DATA_TYPE column. VARIANT_FALSE: The data type is not the best match. For each set of rows in which the value of the DATA_TYPE column is the same, the BEST_MATCH column is set to VARIANT_TRUE in only one row.

The default sort order for the **adSchemaProviderTypes** rowset is DATA_TYPE.

adSchemaTables

The **adSchemaTables** *QueryType* identifies the tables defined in the catalog that are accessible to a given user. This *QueryType* is supported by the Microsoft OLE DB Provider for AS/400 and VSAM and the Microsoft OLE DB Provider for DB2.

The rowset returned by an **adSchemaTables** *QueryType* contains the columns listed in the following table.

Column name	Type indicator	Description
TABLE_CATALOG	DBTYPE_WSTR	Catalog name. NULL if the provider does not support catalogs.
TABLE_SCHEMA	DBTYPE_WSTR	Unqualified Schema Name. NULL if the provider does not support schemas.
TABLE_NAME	DBTYPE_WSTR	Table name.
TABLE_TYPE	DBTYPE_WSTR	Table type. One of the following or a provider-specific value. "ALIAS" "TABLE" "SYNONYM" "SYSTEM TABLE" "VIEW" "GLOBAL TEMPORARY" "LOCAL TEMPORARY"
TABLE_GUID	DBTYPE_GUID	GUID that uniquely identifies the table. Providers that do not use GUIDs to identify tables should return NULL in this column.
DESCRIPTION	DBTYPE_WSTR	Human-readable description of the table.

The default sort order for the **adSchemaTables** rowset is TABLE_TYPE, TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.

OriginalValue Property (ADO)

The **OriginalValue** property on a **Field** object indicates the value of a **Field** that existed in the record before any changes were made. This property returns a Variant.

Syntax

```
oldValue = currentfield.OriginalValue
```

Remarks

The **OriginalValue** property is used to return the original field value for a field from the current record.

In immediate update mode (the provider writes changes to the underlying data source once the **Update** method is called), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **Update** method call). This is the same value that the **CancelUpdate** method uses to replace the **Value** property.

In batch update mode (the provider caches multiple changes and writes them to the underlying data source only when the **UpdateBatch** method is called), the **OriginalValue** property returns the field value that existed prior to any changes (that is, since the last **UpdateBatch** method call). This is the same value that the **CancelBatch** method uses to replace the **Value** property. When this property is used with the **UnderlyingValue** property, you can resolve conflicts that arise from batch updates.

Precision Property (ADO)

The **Precision** property on a **Field** object indicates the degree of precision for Numeric values for numeric **Field** objects. This property returns a byte value indicating the maximum number of digits used to represent numeric values in a **Field** object.

Syntax

```
numericPrecision = currentfield.Precision
```

Remarks

The **Precision** property is used to return the precision of a numeric field object.

The byte value that the **Precision** property will return is dependent on the data type of the **Field** object. The value for the ActiveX® Data Objects (ADO) data type of the **Field** object can be one of the enumerated values for **DataTypeEnum** as listed in the following table.

Enumeration	Value	Description
adEmpty	0	This data type indicates that no value was specified (DBTYPE_EMPTY).
adSmallInt	2	This data type indicates a 2-byte (16-bit) signed integer (DBTYPE_I2).
adInteger	3	This data type indicates a 4-byte (32-bit) signed integer (DBTYPE_I4).
adSingle	4	This data type indicates a 4-byte (32-bit) single precision IEEE floating point number (DBTYPE_R4).
adDouble	5	This data type indicates an 8-byte (64-bit) double precision IEEE floating point number (DBTYPE_R8).
adCurrency	6	A data type indicates a currency value (DBTYPE_CY). Currency is a fixed-point number with 4 digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000. This data type is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adDate	7	This data type indicates a date value stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBSTR	8	This data type indicates a null-terminated Unicode character string (DBTYPE_BSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIDispatch	9	This data type indicates a pointer to an IDispatch interface on an OLE object (DBTYPE_IDISPATCH). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adError	10	This data type indicates a 32-bit error code (DBTYPE_ERROR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBoolean	11	This data type indicates a Boolean value (DBTYPE_BOOL). This data type is not supported by the OLE DB Provider for AS/400 and VSAM.

adVariant	12	This data type indicates an automation variant (DBTYPE_VARIANT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIUnknown	13	This data type indicates a pointer to an IUnknown interface on an OLE object (DBTYPE_IUNKNOWN). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adDecimal	14	This data type indicates numeric data with a fixed precision and scale (DBTYPE_DECIMAL).
adTinyInt	16	This data type indicates a single-byte (8-bit) signed integer (DBTYPE_I1). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedTinyInt	17	This data type indicates a single-byte (8-bit) unsigned integer (DBTYPE_UI1). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedSmallInt	18	This data type indicates a 2-byte (16-bit) unsigned integer (DBTYPE_UI2). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedInt	19	This data type indicates a 4-byte (32-bit) unsigned integer (DBTYPE_UI4). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBigInt	20	This data type indicates an 8-byte (64-bit) signed integer (DBTYPE_I8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM.
adUnsignedBigInt	21	This data type indicates an 8-byte (64-bit) unsigned integer (DBTYPE_UI8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adGUID	72	This data type indicates a globally unique identifier or GUID (DBTYPE_GUID). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBinary	128	This data type indicates fixed-length binary data (DBTYPE_BYTES).
adChar	129	This data type indicates a character string value (DBTYPE_STR).
adWChar	130	This data type indicates a null-terminated Unicode character string (DBTYPE_WSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adNumeric	131	This data type indicates numeric data where the precision and scale are exactly as specified (DBTYPE_NUMERIC).
adUserDefined	132	This data type indicates user-defined data (DBTYPE_UDT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

adDBDate	133	This data type indicates an OLE DB date structure (DBTYPE_DATE).
adDBTime	134	This data type indicates an OLE DB time structure (DBTYPE_TIME).
adDBTimeStamp	135	This data type indicates an OLE DB timestamp structure (DBTYPE_TIMESTAMP).
adVarChar	200	This data type indicates variable-length character data (DBTYPE_STR).
adLongVarChar	201	This data type indicates a long string value.
adVarWChar	202	This data type indicates a Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adLongVarWChar	203	This data type indicates a long Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adVarBinary	204	This data type indicates variable-length binary data (DBTYPE_BYTES).
adLongVarBinary	205	This data type indicates a long binary value.

Note that the **Precision** property returns values that differ from the precision of the host data type for the ActiveX® Data Objects (ADO) data types as listed in the following table.

ADO data type	Comments
adSmallInt	The precision on the host is 4, but the OLE DB Provider returns a precision of 5.
adInteger	The precision on the host is 8, but the OLE DB Provider returns a precision of 10.
adSingle	The precision on the host is 9, but the OLE DB Provider returns a precision of 7.
adDouble	The precision on the host is 17, but the OLE DB Provider returns a precision of 15.

Provider Property (ADO)

The **Provider** property on a **Connection** object indicates the name of the provider. This property sets or returns a String value.

Syntax

```
oldProvider = currentConnection.Provider
currentConnection.Provider = "SNAOLEDB"
```

Remarks

The **Provider** property is used to set or return the name of the provider for the connection. This property can also be set by the contents of **ConnectionString** property or the **ConnectionString** argument of the **Open** method. However, specifying a provider in more than one place while calling the **Open** method can have unpredictable results. If no provider is specified, the property will default to MSDASQL (Microsoft® OLE DB Provider for ODBC).

The Microsoft OLE DB Provider for AS/400 and VSAM requires "SNAOLEDB" as the **Provider** property string.

The Microsoft OLE DB Provider for DB2 requires "DB2OLEDB" as the **Provider** property string.

The **Provider** property is read/write when the connection is closed and read-only when it is open. The setting does not take effect until either the **Connection** object is opened or the **Properties** collection of the **Connection** object is accessed. If the setting is invalid, an error occurs.

Refresh Method (ADO)

The **Refresh** method on a **Collection** object updates the objects in a collection to reflect objects available from and specific to the OLE DB provider.

Syntax

```
collection.Refresh
```

Parameters

None.

Remarks

This method is only supported on the **Field**s and **Properties** collections under the Microsoft® OLE DB Provider for AS/400 and VSAM.

The **Refresh** method accomplishes different tasks depending on the collection object on which it is called.

Using the **Refresh** method on the **Field**s collection has no visible effect. To retrieve changes from the underlying database structure, either the **Requery** method must be used or, if the **Recordset** object does not support bookmarks, the **MoveFirst** method must be used.

Using the **Refresh** method on a **Properties** collection of some objects populates the collection with the dynamic properties the provider exposes. These properties provide information about features specific to the provider beyond the built-in properties ActiveX® Data Objects (ADO) supports. The OLE DB Data Provider for AS/400 and VSAM does not support any provider-specific properties.

Requery Method (ADO)

The **Requery** method on a **Recordset** object updates the data in a **Recordset** object by re-executing the query on which the object is based.

Syntax

```
recordset.Requery
```

Parameters

None.

Remarks

The **Requery** method is used to refresh the entire contents of a **Recordset** object from the data source by reissuing the original command and retrieving the data a second time. Calling this method is equivalent to calling the **Close** and **Open** methods in succession. If you are editing the current record or adding a new record, an error occurs.

While the **Recordset** object is open, the properties that define the nature of the cursor (**CursorType**, **LockType**, **MaxRecords**, and other properties) are read-only. Thus, the **Requery** method can only refresh the current cursor. To change any of the cursor properties and view the results, the **Close** method must be used so that the properties become read/write again. You can then change the property settings and call the **Open** method to reopen the cursor.

Save Method (ADO)

The **Save** method on a **Recordset** object saves the Recordset in a file or Stream object.

Syntax

```
recordset.Save Destination, Persistent Format
```

Parameters

Destination

This optional parameter specifies a Variant representing the complete path name of the file where the **Recordset** is to be saved, or a reference to a **Stream** object.

Persistent Format

This optional parameter specifies a Long integer value representing a **PersistFormatEnum** value that specifies the format in which the **Recordset** is to be saved (XML or ADTG). The default value is **adPersistADTG**.

The **PersistFormatEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adPersistADTG	0	This value indicates Microsoft Advanced Data TableGram (ADTG) format.
adPersistXML	1	This value indicates Extensible Markup Language (XML) format.

Remarks

The **Save** method can only be invoked on an open **Recordset**. Use the **Open** method to later restore the **Recordset** from *Destination*.

When using the Microsoft® OLE DB Provider for AS/400 and VSAM and the Filter property is in effect for the **Recordset**, then only the rows accessible under the filter are saved.

The first time you save the **Recordset**, it is optional to specify *Destination*. If the *Destination* parameter is omitted, a new file will be created with a name set to the value of the Source Property of the **Recordset**. See the topic Source Property on a Recordset Object (ADO) for more information.

The *Destination* parameter should be omitted when you subsequently call **Save** after the first save, or a run-time error will occur. If you subsequently call **Save** with a new *Destination*, the **Recordset** is saved to the new destination. However, the new destination and the original destination will both be open.

Save does not close the **Recordset** or *Destination*, so you can continue to work with the **Recordset** and save your most recent changes. *Destination* remains open until the **Recordset** is closed, during which time other applications can read but not write to *Destination*.

For reasons of security, the **Save** method permits only the use of low and custom security settings from a script executed by Microsoft Internet Explorer. For a more detailed explanation of security issues, see "ADO and RDS Security Issues in Microsoft Internet Explorer" found in the ActiveX® Data Objects (ADO) Technical Articles of the Microsoft Data Access Technical Articles.

If the **Save** method is called while an asynchronous **Recordset** fetch, execute, or update operation is in progress, then **Save** waits until the asynchronous operation is complete.

Records are saved beginning with the first row of the **Recordset**. When the **Save** method is finished, the current row position is moved to the first row of the **Recordset**.

For best results, set the CursorLocation Property property to **adUseClient** with **Save**. If your provider does not support all of the features necessary to save **Recordset** objects, the Cursor Service will provide these features.

When a **Recordset** is persisted with the **CursorLocation** property set to **adUseServer**, the update capability for the **Recordset** is limited. Typically, only single-table updates, insertions, and deletions are allowed (dependent on features supported by the provider). The **Resync** method is also unavailable in this configuration.

Note that saving a **Recordset** with **Fields** of type **adVariant**, **adIDispatch**, or **adIUnknown** is not supported by ADO and can cause unpredictable results.

Because the *Destination* parameter can accept any object that supports the OLE DB IStream interface, a **Recordset** can be saved directly to the ASP Response object. For more information, see the XML Recordset Persistence Scenario in the *ADO Programmer's Reference*.

Sort Property (ADO)

The **Sort** property on a **Recordset** object indicates that a recordset should be sorted.

Syntax

```
Recordset.Sort BSTR Criteria
```

Parameters

Criteria

This parameter specifies the criteria used for sorting the **Recordset** object. This **Sort** property contains a comma-delimited list of column names and a direction specifier (ascending or descending) to be used for sorting records in a **Recordset** object. The direction specifier is a string (ASC or DESC). When a direction is not specified, the direction defaults to ascending.

An example of a **Sort** property criteria is as follows:

"LastName ASC, FirstName DESC, Initial"

Remarks

The **Sort** property is not supported by the OLE DB Provider for DB2 or the ODBC Driver for DB2.

The **Sort** property is used with an open **Recordset** object based on an AS/400 physical file. The **Sort** property allows the user to indicate which logical view to apply to an AS/400 physical file. The logical view must be a valid index specified in the description of the AS/400 physical file. The logical view is provided by the AS/400 logical file. The Microsoft® OLE DB Provider for AS/400 and VSAM responds to a **Sort** request by first closing the open physical file, and then opening the logical file that points back to the data in the physical file.

The **Recordset Sort** property is only supported on AS/400 hosts. If the user opens a **Recordset** object based on an AS/400 logical file, then there is likely no need to use **Recordset**.**Sort**. For performance reasons, applications should be written to open the AS/400 logical file first, because the overhead is so much greater when opening a physical file first.

If the CursorLocation Property property is set to **adUseClient** (use the client cursor engine), the **Sort** property will work if MDAC 2.0 or later is installed but will not work properly with earlier versions of ADO.

Source Property on an Error Object (ADO)

The **Source** property on an **Error** object indicates the name of the object or application that originally generated an error. This property returns a String value that indicates the name of an object or application.

Syntax

```
errorSource = currentError.Source
```

Remarks

The **Source** property on an **Error** object is used to determine the name of the object or application that originally generated an error. This could be the object's class name or programmatic ID.

For errors in ADO, the property value will be **ADODB.***ObjectName*, where *ObjectName* is the name of the object that triggered the error. For ADOX and ADO MD, the value will be **ADOX.***ObjectName* and **ADOMD.***ObjectName*, respectively.

Based on the error documentation from the **Source**, Number Property (ADO), and Description Property (ADO) for **Error** objects, you can write code that will handle the error appropriately.

The **Source** property is read-only for **Error** objects.

Source Property on a Recordset Object (ADO)

The **Source** property on a **Recordset** object indicates the data source for the Recordset object. This property sets a String value or **Command** object reference or returns only a String value that indicates the source of the **Recordset**.

Syntax

```
currentSource = currentRecordset.Source
recordset.Source = newSource
```

Remarks

The **Source** property on a Recordset is used to specify a data source for a **Recordset** object.

Using the Microsoft® OLE DB Provider for AS/400 and VSAM, the **Source** property can be either a **Command** object variable or a table name.

Using the Microsoft OLE DB Provider for DB2, the Source property can be one of the following: a **Command** object variable, an SQL statement, or a stored procedure. If the **Source** property is an SQL statement or a stored procedure, you can optimize performance by passing the appropriate *Options* argument with the **Open** method call.

If you set the **Source** property to a **Command** object, the **ActiveConnection** property of the **Recordset** object will inherit the value of the **ActiveConnection** property for the specified **Command** object. However, reading the **Source** property does not return a **Command** object; instead, it returns the **CommandText** property of the **Command** object to which you set the **Source** property.

The **Source** property is read/write for closed **Recordset** objects and read-only for open **Recordset** objects.

State Property (ADO)

The **State** property on a **Connection**, **Command**, or **Recordset** object describes the current state of an object. This property sets or returns a Long value.

Syntax

```
oldState = currentConnection.State
currentConnection.State = adStateClosed
```

Remarks

The **State** property is used to set or return the current state of an object. The value of the **State** property can be one of the enumerated values listed in the following table.

Enumeration	Value	Description
adStateClosed	0	This value indicates that the object is closed. This is the default value.
adStateOpen	1	This value indicates that the object is open.

The **State** property can be used to determine the current state of a given object at any time.

Status Property (ADO)

The **Status** property on a **Recordset** object indicates the status of the current record with respect to batch updates or other bulk operations. This property returns a Long value.

Syntax

```
oldStatus = currentRecordset.Status
```

Remarks

The **Status** property is used to return the current status of a recordset object at any time. The value of the **Status** property returns a sum of the **RecordStatusEnum** enumerated values listed in the following table.

Enumeration	Value	Description
adRecOK	0	This value indicates that the recordset object was successfully updated.
adRecNew	0x1	This value indicates that the recordset object is new.
adRecModified	0x2	This value indicates that the recordset object was modified.
adRecDeleted	0x4	This value indicates that the recordset object was deleted.
adRecUnmodified	0x8	This value indicates that the recordset object was not modified.
adRecInvalid	0x10	This value indicates that the recordset object was not saved because its bookmark is invalid.
adRecMultipleChanges	0x40	This value indicates that the recordset object was not saved because it would have affected multiple records.
adRecPendingChanges	0x80	This value indicates that the recordset object was not saved because it refers to a pending insert.
adRecCanceled	0x100	This value indicates that the recordset object was not saved because the operation was canceled.
adRecCantRelease	0x400	This value indicates that the new recordset object was not saved because of existing record locks.
adRecConcurencyViolation	0x800	This value indicates that the recordset object was not saved because optimistic concurrency was in use.
adRecIntegrityViolation	0x1000	This value indicates that the recordset object was not saved because the user violated integrity constraints.
adRecMaxChangesExceeded	0x2000	This value indicates that the recordset object was not saved because there were too many pending changes.
adRecObjectOpen	0x4000	This value indicates that the recordset object was not saved because of a conflict with an open storage object.
adRecOutOfMemory	0x8000	This value indicates that the recordset object was not saved because the computer has run out of memory.

adRecPermissionDenied	0x10000	This value indicates that the recordset object was not saved because the user has insufficient permissions.
adRecSchemaViolation	0x20000	This value indicates that the recordset object was not saved because it violates the structure of the underlying database.
adRecDBDeleted	0x40000	This value indicates that the recordset object has already been deleted from the data source.

Use the **Status** property to see what changes are pending for records modified during batch updating. You can also use the **Status** property to view the status of records that fail during bulk operations such as when you call the **Resync**, **UpdateBatch**, or **CancelBatch** methods on a recordset object, or set the **Filter** property on a recordset object to an array of bookmarks. With this property, you can determine how a given record failed and resolve it accordingly.

Supports Method (ADO)

The **Supports** method on a **Recordset** object determines whether a specified **Recordset** object supports a particular type of feature.

Syntax

```
boolean = recordset.Supports ( CursorOptions )
```

Parameters

CursorOptions

This parameter specifies a Long expression that consists of one or more of the **CursorOptionEnum** values indicating which feature is being queried.

The **CursorOptionEnum** value can be one of the constants listed in the table following the Parameters section.

Values for CursorOptions

Enumeration	Value	Description
adAddNew	0x1000400	This value indicates whether the **AddNew** method can be used to add new records.
adApproxPosition	0x4000	This value indicates whether the **AbsolutePosition** and **AbsolutePage** properties can read and set.
adBookMark	0x2000	This value indicates whether the **Bookmark** property can be used to access specific records.
adDelete	0x1000800	This value indicates whether the **Delete** method can be used to delete records.
adFind	0x80000	This value indicates whether the **Find** method can be used to locate a row in a **Recordset**.
adHoldRecords	0x100	This value indicates whether you can retrieve more records or change the next retrieve position without committing all pending changes.
adIndex	0x100000	This value indicates whether the **Index** property can be name an index.
adMovePrevious	0x200	This value indicates whether the **MoveFirst** and **MovePrevious** methods, and **Move** or **GetRows** methods can be used to move the current record position backward without requiring bookmarks.
adNotify	0x40000	This value indicates that the underlying data provider supports notifications (which determines whether **Recordset** events are supported).
adResync	0x20000	This value indicates whether the recordset cursor can be updated with the data visible in the underlying database using the **Resync** method.
adSeek	0x200000	This value indicates whether the **Seek** method can be used to locate a row in a **Recordset**.

adUpdate	0x10080000	This value indicates whether the **Update** method can be used to modify existing data.
adUpdateBatch	0x10000	This value indicates whether batch updating can be used on the recordset (the **UpdateBatch** and **CancelBatch** methods) to transmit changes to the provider in groups.

Return Value

Returns a Boolean value that indicates whether all of the features identified by the *CursorOptions* argument are supported by the provider.

Remarks

The **Supports** method is used to determine what types of features (methods and properties) a **Recordset** object supports. If the **Recordset** object supports the features whose corresponding constants are in *CursorOptions*, the **Supports** method returns **True**. Otherwise, it returns **False**.

Although the **Supports** method may return **True** for a given feature, it does not guarantee that the OLE DB Provider can make the feature available under all circumstances. The **Supports** method simply returns whether the provider can support the specified function assuming certain conditions are met. For example, the **Supports** method may indicate that a **Recordset** object supports updates even though the cursor is based on a multi-table join, some columns of which are not updatable.

Type Property (ADO)

The **Type** property on a **Field** object indicates the operational type or data type for **Field** or **Property** objects. This property sets or returns a **DataTypeEnum** value.

Syntax

```
datatype = currentfield.Type
```

Remarks

The **Type** property is used to return the data type of a numeric field object.

The value returned by the **Type** property on a **Field** object can be one of the enumerated values for **DataTypeEnum** listed in the following table.

Enumeration	Value	Description
adEmpty	0	This data type indicates that no value was specified (DBTYPE_EMPTY).
adSmallInt	2	This data type indicates a 2-byte (16-bit) signed integer (DBTYPE_I2).
adInteger	3	This data type indicates a 4-byte (32bit) signed integer (DBTYPE_I4).
adSingle	4	This data type indicates a 4-byte (32-bit) single-precision IEEE floating-point number (DBTYPE_R4).
adDouble	5	This data type indicates an 8-byte (64-bit) double-precision IEEE floating-point number (DBTYPE_R8).
adCurrency	6	A data type indicates a currency value (DBTYPE_CY). Currency is a fixed-point number with four digits to the right of the decimal point. It is stored in an 8-byte signed integer scaled by 10,000. This data type is not supported by the Microsoft® OLE DB Provider for AS/400 and VSAM or the Microsoft OLE DB Provider for DB2.
adDate	7	This data type indicates a date value stored as a Double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBSTR	8	This data type indicates a null-terminated Unicode character string (DBTYPE_BSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIDispatch	9	This data type indicates a pointer to an **IDispatch** interface on an OLE object (DBTYPE_IDISPATCH). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adError	10	This data type indicates a 32-bit error code (DBTYPE_ERROR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBoolean	11	This data type indicates a Boolean value (DBTYPE_BOOL). This data type is not supported by the OLE DB Provider for AS/400 and VSAM.

adVariant	12	This data type indicates an Automation variant (DBTYPE_VARIANT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adIUnknown	13	This data type indicates a pointer to an **IUnknown** interface on an OLE object (DBTYPE_IUNKNOWN). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adDecimal	14	This data type indicates numeric data with a fixed precision and scale (DBTYPE_DECIMAL).
adTinyInt	16	This data type indicates a single-byte (8-bit) signed integer (DBTYPE_I1). This data type is not supported by the OLE DB Provider.
adUnsignedTinyInt	17	This data type indicates a single-byte (8-bit) unsigned integer (DBTYPE_UI1). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedSmallInt	18	This data type indicates a 2-byte (16-bit) unsigned integer (DBTYPE_UI2). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adUnsignedInt	19	This data type indicates a 4-byte (32-bit) unsigned integer (DBTYPE_UI4). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBigInt	20	This data type indicates an 8-byte (64-bit) signed integer (DBTYPE_I8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM.
adUnsignedBigInt	21	This data type indicates an 8-byte (64-bit) unsigned integer (DBTYPE_UI8). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adGUID	72	This data type indicates a globally unique identifier or GUID (DBTYPE_GUID). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adBinary	128	This data type indicates fixed-length binary data (DBTYPE_BYTES).
adChar	129	This data type indicates a character string value (DBTYPE_STR).
adWChar	130	This data type indicates a null-terminated Unicode character string (DBTYPE_WSTR). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adNumeric	131	This data type indicates numeric data where the precision and scale are exactly as specified (DBTYPE_NUMERIC).
adUserDefined	132	This data type indicates user-defined data (DBTYPE_UDT). This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.

adDBDate	133	This data type indicates an OLE DB date structure (DBTYPE_DATE).
adDBTime	134	This data type indicates an OLE DB time structure (DBTYPE_TIME).
adDBTimeStamp	135	This data type indicates an OLE DB timestamp structure (DBTYPE_TIMESTAMP).
adVarChar	200	This data type indicates variable-length character data (DBTYPE_STR).
adLongVarChar	201	This data type indicates a long string value.
adVarWChar	202	This data type indicates a Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adLongVarWChar	203	This data type indicates a long Unicode string value. This data type is not supported by the OLE DB Provider for AS/400 and VSAM or the OLE DB Provider for DB2.
adVarBinary	204	This data type indicates variable-length binary data (DBTYPE_BYTES).
adLongVarBinary	205	This data type indicates a long binary value.

The corresponding OLE DB type indicator is shown in parentheses in the description column of the above table. For more information on OLE DB data types, see the *OLE DB 2.0 Programmer's Reference*.

UnderlyingValue Property (ADO)

The **UnderlyingValue** property on a **Field** object indicates the **Field** object's current value in the database. This property returns a Variant.

Syntax

```
actualValue = currentfield.UnderlyingValue
```

Remarks

The **UnderlyingValue** property is used to return the current field value from the database. The field value in the **UnderlyingValue** property is the value that is visible to your transaction and may be the result of a recent update by another transaction. This may differ from the **OriginalValue** property, which reflects the value that was originally returned to the **Recordset**.

This is similar to the affect of calling the **Resync** method, however the **UnderlyingValue** property returns only the value for a specific field from the current record. This is the same value that the **Resync** method uses to replace the **Value** property.

When this property is used with the **OriginalValue** property, you can resolve conflicts that arise from batch updates.

Update Method (ADO)

The **Update** method on a **Recordset** object saves any changes you make to the current record of a **Recordset** object.

Syntax

```
recordset.Update Fields, Values
```

Parameters

Fields

This optional parameter specifies a Variant representing a single name or a Variant array representing names or ordinal positions of the field or fields you want to modify.

Values

This optional parameter specifies a Variant representing a single value or a Variant array representing values for the field or fields in the new record.

Remarks

The **Update** method is used to save any changes you make to the current record of a **Recordset** object since calling the **AddNew** method or since changing any field values in an existing record. The **Recordset** object must support updates for the **Update** method to be used successfully.

To set field values, do one of the following:

- Assign values to a **Field** object's **Value** property and call the **Update** method.
- Pass a field name and a value as arguments with the **Update** call.
- Pass an array of field names and an array of values with the **Update** call.

When arrays of fields and values are used, there must be an equal number of elements in both arrays. Also, the order of field names must match the order of field values. If the number and order of fields and values do not match, an error occurs.

If the **Recordset** object supports batch updating, then multiple changes to one or more records can be cached locally when the **UpdateBatch** method is called. If you are editing the current record or adding a new record when the **UpdateBatch** method is called, ActiveX® Data Objects (ADO) will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the OLE DB Provider.

If you move from the record you are adding or editing before calling the **Update** method, ADO will automatically call **Update** to save the changes. The **CancelUpdate** method must be called if you want to cancel any changes made to the current record or to discard a newly added record.

The current record remains current after the **Update** method is called.

UpdateBatch Method (ADO)

The **UpdateBatch** method on a **Recordset** object writes all pending batch updates to the host.

Syntax

```
recordset.UpdateBatch AffectedRecords
```

Parameters

AffectedRecords

This optional parameter specifies an **AffectEnum** value that determines how many records the **UpdateBatch** method will affect.

The **AffectEnum** value can be one of the constants listed in the following table.

Enumeration	Value	Description
adAffectCurrent	1	This value writes pending changes only for the current record.
adAffectGroup	2	This value writes pending changes for the records that satisfy the current Filter property setting. You must set the **Filter** property to one of the valid predefined constants to use this option.
adAffectAll	3	This value writes pending changes for all the records in the **Recordset** object, including any hidden by the current **Filter** property setting. This value is the default.

Remarks

The **UpdateBatch** method is used when modifying a **Recordset** object in batch update mode to transmit all changes made in a **Recordset** object to the underlying database.

If the **Recordset** object supports batch updating, then multiple changes to one or more records can be cached locally until the **UpdateBatch** method is called. If you are editing the current record or adding a new record when the **UpdateBatch** method is called, ADO will automatically call the **Update** method to save any pending changes to the current record before transmitting the batched changes to the provider.

If the attempt to transmit changes fails because of a conflict with the underlying data (for example, a record has already been deleted by another user), the provider returns warnings to the **Errors** collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the **Filter** property (**adFilterAffectedRecords**) and the **Status** property to locate records with conflicts.

To cancel all pending batch updates, use the **CancelBatch** method.

Value Property (ADO)

The **Value** property on a **Field** object indicates the value assigned to a **Field** or **Property** object. This property sets or returns a Variant value. The default value depends on the **Type** property of the **Field** object.

Syntax

```
oldValue = currentfield.Value
currentField.Value = newValue
```

Remarks

The **Value** property is used to set or return data from **Field** objects or to set or return property settings with **Property** objects. Whether the **Value** property is read/write or read-only depends upon numerous factors. For a **Field** object, this includes whether the **Recordset** was opened as read-only or read/write.

ActiveX® Data Objects (ADO) allows setting and returning long binary data with the **Value** property from the database.

Version Property (ADO)

The **Version** property on a **Connection** object indicates the ActiveX® Data Objects (ADO) version number. This property returns a String value.

Syntax

```
versionADO = currentConnection.Version
```

Remarks

The **Version** property is used to return the version number of the ADO implementation. The version of the provider will be available as a dynamic property in the **Properties** collection.

Network Integration Programmer's Reference

This section of Host Integration Server 2009 Help describes the objects, methods, properties, controls, and other interfaces that enable you to integrate Host Integration Server network technologies into your application.

In This Section

APPC Programmer's Reference

CPI-C Programmer's Reference

LUA Programmer's Reference

3270 Emulation Programmer's Reference

SNA Internationalization Programmer's Reference

SNA Print Server Data Filter Programmer's Reference

Session Integrator Programmer's Reference

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Reference

Reference

SNADIS Drivers Programmer's Reference

Related Sections

Network Integration Programmer's Guide

Network Integration Samples

See Also

Other Resources

Programmer's Reference

APPC Programmer's Reference

This section of Host Integration Server 2009 Help provides information about the verbs, extensions, and return codes that make up the APPC programming interface.

For general information about programming for APPC, see the APPC Programmer's Guide section of the SDK.

For sample code using APPC, see APPC Samples.

In This Section

APPC Management Verbs

This section describes the Advanced Program-to-Program Communications (APPC) management verbs. The management verbs enable you to establish APPC LU 6.2 session limits, obtain configuration information and current operating values for the SNA node, and activate or deactivate sessions. The description of each verb provides:

- A definition of the verb.
- The structure defining the verb control block (VCB) used by the verb. The structure is contained in the WINAPPC.H file. The length of each VCB field is in bytes. Fields beginning with reserv (for example, reserv2) are reserved.
- The parameters (VCB fields) supplied to and returned by APPC. A description of each parameter is provided, along with its possible values and other information.
- The conversation state(s) in which the verb can be issued.
- The state(s) to which the conversation can change upon return from the verb. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the verb.

Most parameters supplied to and returned by APPC are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants, which are established by **#define** statements in the WINAPPC.H header file. For example, the **opcode** (operation code) member of the **mc_send_data** structure used by the **MC_SEND_DATA** verb is the hexadecimal value represented by the symbolic constant AP_M_SEND_DATA. Use only the symbolic constants when writing transaction programs (TPs).

In This Section

ACTIVATE_SESSION

The **ACTIVATE_SESSION** verb requests Microsoft® Host Integration Server to activate a session between the local logical unit (LU) and a specified partner LU, using a specified mode. This verb completes either when the specified session has become active or when it has failed.

The following structure describes the verb control block used by the **ACTIVATE_SESSION** verb.

Syntax

```
typedef struct activate_session {
    unsigned short  opcode;
    unsigned char   reserv2[2];
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   reserv3[8];
    unsigned char   lu_alias[8];
    unsigned char   plu_alias[8];
    unsigned char   mode_name[8];
    unsigned char   fqplu_name[17];
    unsigned char   polarity;
    unsigned char   session_id[8];
    unsigned long   conv_group_id;
    unsigned char   reserv4[1];
    unsigned char   type;
    HANDLE          deactivation_event;
    unsigned short* p_deactivation_status;
    unsigned char   reserv5[10];
} ACTIVATE_SESSION;
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_ACTIVATE_SESSION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

reserv3

A reserved field.

lu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the LU. If the default local LU is to be used, fill this parameter with spaces.

plu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the partner LU. If the default remote LU is to be used, fill this parameter with spaces. If the partner LU is to be specified with the **fqplu_name** parameter, fill this parameter with binary zeros.

mode_name

Supplied parameter. Specifies the EBCDIC (type A) mode name.

fqplu_name

Supplied parameter. Provides the partner LU name in EBCDIC (type A) when no **plu_alias** name is defined at the local node and the partner LU is located at a different node. This parameter is ignored if **plu_alias** is specified.

polarity

Supplied parameter. Specifies the polarity for the session. The possible values are:

AP_POL_EITHER

If AP_POL_EITHER is set, ACTIVATE_SESSION activates a first speaker session if available; otherwise a bidder session is activated.

AP_POL_FIRST_SPEAKER

If AP_POL_FIRST_SPEAKER is set, ACTIVATE_SESSION only succeeds if a session of the requested polarity is available.

AP_POL_BIDDER

If AP_POL_BIDDER is set, ACTIVATE_SESSION only succeeds if a session of the requested polarity is available.

session_id

Returned parameter. Provides the 8-byte identifier of the activate session.

conv_group_id

Returned parameter. Provides the conversation group identifier. This parameter can be specified on **ALLOCATE** and **MC_ALLOCATE** verbs to start conversations on this particular session.

reserv4

A reserved field.

type

Supplied parameter. Specifies the type of activation. Possible values are:

AP_ACT_ACTIVE

If AP_ACT_ACTIVE is specified, then Host Integration Server will attempt to start the required session (by sending the BIND or INIT-SELF).

AP_ACT_PASSIVE

If AP_ACT_PASSIVE is specified, then Host Integration Server will not attempt to start the session and the verb will complete when the partner has started the session.

deactivation_event

Supplied parameter. Provides an event handle that APPC is to signal when the session is deactivated. The event handle should be obtained by calling either the **CreateEvent** or **OpenEvent** Win32® function.

p_deactivation_status

Returned parameter. A pointer to a value that is set when the deactivation event is signaled to provide completion status. The following values can be returned.

AP_SESSION_DEACTIVATED

AP_COMM_SUBSYSTEM_ABENDED

reserv5

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully. The secondary return code indicates the polarity of the established session. The following values can be returned.

AP_POL_FIRST_SPEAKER

AP_POL_BIDDER

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_INVALID_LU_ALIAS

Secondary return code; APPC cannot find the specified **lu_alias** among those defined.

AP_INVALID_PLU_ALIAS

Secondary return code; APPC did not recognize the specified **plu_alias**.

AP_INVALID_MODE_NAME

Secondary return code; APPC did not recognize the specified **mode_name**.

AP_INVALID_FQPLU_NAME

Secondary return code; APPC did not recognize the specified **fqplu_name**.

AP_INVALID_POLARITY

Secondary return code; APPC did not recognize the specified **polarity**.

AP_INVALID_TYPE

Secondary return code; APPC did not recognize the specified **type**.

AP_ACTIVATION_FAIL_NO_RETRY

Primary return code; the session could not be activated because of a condition that requires action (such as a configuration mismatch or a session protocol error).

AP_ACTIVATION_FAIL_RETRY

Primary return code; the session could not be activated because of a temporary condition (such as a link failure).

AP_SESSION_LIMITS_EXCEEDED

Primary return code; the session could not be activated because the session limits have been exceeded.

AP_SESSION_LIMITS_CLOSED

Primary return code; the session could not be activated because the session limits are closed (that is, zero).

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions occurred:

The node used by this conversation encountered an ABEND.

The connection between the TP and the PU 2.1 node has been broken (local area network error occurred).

The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

Remarks

This verb supports both active and passive activation.

The active form of this verb results in Host Integration Server trying to initiate the session (by sending a BIND for independent LUs or an INIT-SELF for dependent LUs). The active form of this verb will also result in the following behavior:

- If the connection to the partner LU is inactive and is configured as on-demand, the Node will attempt to start the

connection.

- If dynamic partnering is being used, the Node will set up the LU-LU/MODE partnership.
- If CNOS has not run, the Node will start CNOS (but will not change any of the session limits).

The passive form does not attempt to start the session, but completes when the LU is started by a BIND from its partner LU. For independent LUs, multiple passive **ACTIVATE_SESSION** verbs can be queued up for the same LU-LU/MODE, and complete in turn as new sessions are started.

This verb also includes a deactivation event, which is posted when the session is deactivated by any method other than a **DEACTIVATE_SESSION** verb (for example, an unsolicited UNBIND from its partner LU results in this event being posted).

CNOS

The **CNOS** (Change Number of Sessions) verb establishes APPC LU 6.2 session limits.

The following structure describes the verb control block used by the **CNOS** verb.

Syntax

```
typedef struct cnos {
    unsigned short  opcode;
    unsigned char   reserv2[2];
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   key[8];
    unsigned char   lu_alias[8];
    unsigned char   plu_alias[8];
    unsigned char   fqplu_name[17];
    unsigned char   reserv3;
    unsigned char   mode_name[8];
    unsigned int    mode_name_select:1;
    unsigned int    set_negotiable:1;
    unsigned int    reserv4:6;
    unsigned int    reserv5:8;
    unsigned short  plu_mode_sess_lim;
    unsigned short  min_conwinners_source;
    unsigned short  min_conwinners_target;
    unsigned short  auto_act;
    unsigned int    drain_target:1;
    unsigned int    drain_source:1;
    unsigned int    responsible:1;
    unsigned int    reserv6:5;
    unsigned int    reserv7:8;
} CNOS;
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_CNOS.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

key

Supplied parameter. Specifies either the master or service key in ASCII, if the keylock feature has been secured.

lu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the LU.

plu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the partner LU.

fqplu_name

Supplied parameter. Provides the partner logical unit (LU) name in EBCDIC (type A) when no **plu_alias** name is defined at the local node and the partner LU is located at a different node.

mode_name

Supplied parameter. Specifies the EBCDIC (type A) mode name to be used when the value of **mode_name_select** is AP_ONE.

mode_name_select

Supplied parameter. Specifies the mode name select for which your program is setting or resetting the session limits and contention-winner polarities. Allowed values are AP_ALL or AP_ONE.

set_negotiable

Supplied parameter. Specifies whether APPC is to change the current setting for the maximum negotiable session limit. Allowed values are AP_YES and AP_NO.

reserv4

A 6-bit reserved field.

reserv5

An 8-bit reserved field.

plu_mode_sess_lim

Supplied parameter. Specifies the session limit when the value for **set_negotiable** is YES. Allowed values are 0 to 32767.

min_conwinners_source

Supplied parameter. Specifies the number of sessions of which the LU is guaranteed to be the contention winner. Allowed values are 0 to 32767.

min_conwinners_target

Supplied parameter. Specifies the minimum number of sessions of which the target LU is guaranteed to be the contention winner. Allowed values are 0 to 32767.

auto_act

Supplied parameter. Specifies the number of the local LUs contention-winner sessions for APPC to activate automatically. Allowed values are 0 to 32767. See the Remarks section of this topic before using this parameter.

drain_target

Supplied parameter. Specifies whether the target LU can drain its waiting (outbound) allocation requests. Allowed values are AP_YES and AP_NO.

drain_source

Supplied parameter. Specifies whether the source LU can drain its waiting (outbound) allocation requests. Allowed values are AP_YES and AP_NO.

responsible

Supplied parameter. Specifies which LU is responsible for deactivating the sessions as a result of resetting the session limit for parallel-session connections. Allowed values are AP_SOURCE and AP_TARGET.

reserv6

A 5-bit reserved field.

reserv7

An 8-bit reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_CNOS_ACCEPTED

Secondary return code; APPC accepts the session limits and responsibility as specified.

AP_CNOS_NEGOTIATED

Secondary return code; APPC accepts the session limits and responsibility as negotiable by the partner LU. Values that can be

Secondary return code; APPC accepts the session limits and responsibility as negotiable by the partner LU. Values that can be negotiated are:

plu_mode_session_limit

min_conwinners_source

min_conwinners_target

responsible

drain_target

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE or MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CNOS_LOCAL_RACE_REJECT

Primary return code; APPC is currently processing a **CNOS** verb issued by a local LU.

AP_CNOS_PARTNER_LU_REJECT

Primary return code; the partner LU rejected a **CNOS** request from the local LU.

AP_CNOS_MODE_CLOSED

Secondary return code; the local LU cannot negotiate a nonzero session limit because the local maximum session limit at the partner LU is zero.

AP_CNOS_MODE_NAME_REJECT

Secondary return code; the partner LU does not recognize the specified mode name.

AP_CNOS_COMMAND_RACE_REJECT

Secondary return code; the local LU is currently processing a **CNOS** verb issued by the partner LU.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation encountered an ABEND.

The connection between the transaction program (TP) and the PU 2.1 node has been broken (a local area network error).

The SnaBase at the TPs computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_KEY

Primary return code; the supplied key was incorrect.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_ALL_MODE_MUST_RESET

Secondary return code; APPC does not permit a nonzero session limit when the **mode_name_select** parameter indicates AP_ALL.

AP_AUTOACT_EXCEEDS_SESSLIM

Secondary return code; on the **CNOS** verb, the value for **auto_act** is greater than the value for **plu_mode_sess_lim**.

AP_BAD_LU_ALIAS

Secondary return code; APPC cannot find the specified **lu_alias** among those defined.

AP_BAD_PARTNER_LU_ALIAS

Secondary return code; APPC did not recognize the supplied **plu_alias**.

AP_BAD_SNASVCMG_LIMITS

Secondary return code; your program specified invalid settings for **plu_mode_sess_lim**, **min_conwinners_source**, or **min_conwinners_target** when **mode_name** was supplied.

AP_CHANGE_SRC_DRAINS

Secondary return code; APPC does not permit **mode_name_select** (ONE) and **drain_source** (YES) when **drain_source** (NO) is currently in effect for the specified mode.

AP_CNOS_IMPLICIT_PARALLEL

Secondary return code; APPC does not permit a program to change the session limit for a mode other than the SNASVCMG mode for the implicit partner template when the template specifies parallel sessions. (The term "template" is used because many of the actual values are yet to be filled in.)

AP_CPSVCMG_MODE_NOT_ALLOWED

Secondary return code; the mode named CPSVCMG cannot be specified as the **mode_name** on the deactivate session verb.

AP_EXCEEDS_MAX_ALLOWED

Secondary return code; your program issued a **CNOS** verb, specifying a **plu_mode_sess_lim** number and **set_negotiable** (AP_NO).

AP_MIN_GT_TOTAL

Secondary return code; the sum of **min_conwinners_source** and **min_conwinners_target** specifies a number greater than **plu_mode_sess_lim**.

AP_MODE_CLOSED

Secondary return code; the local LU cannot negotiate a nonzero session limit because the local maximum session limit at the partner LU is zero.

AP_RESET_SNA_DRAINS

Secondary return code; SNASVCMG does not support the drain parameter values.

AP_SINGLE_NOT_SRC_RESP

Secondary return code; for a single-session **CNOS** verb, APPC permits only the local (source) LU to be responsible for deactivating sessions.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_CANT_RAISE_LIMITS

Secondary return code; APPC does not permit setting session limits to a nonzero value unless the limits currently are zero.

AP_LU_DETACHED

Secondary return code; a command has reset the definition of the local LU before **CNOS** tried to specify the LU.

AP_SNASVCMG_RESET_NOT_ALLOWED

Secondary return code; your local program attempted to issue the **CNOS** verb for the mode named SNASVCMG, specifying a session limit of zero.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC verb from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

CNOS identifies an LU by alias alone. If the same local LU alias is used multiple times in a domain (for backup or other purposes) and that LU alias is specified through **CNOS**, the verb can flow to a different LU than the one intended.

If CNOS is not issued to set the mode session limit before a program issues its first APPC ALLOCATE, MC_ALLOCATE, SEND_CONVERSATION, or MC_SEND_CONVERSATION, or Common Programming Interface for Communications (CPI-C) Allocate call for a given partner LU and mode, APPC will internally generate a session limit using the value from the mode definition.

When setting the limits for a parallel-session connection, the two LUs negotiate the mode session limits, drain settings, and responsibility values. APPC updates these parameters in **CNOS** to reflect the settings agreed to by both LUs during negotiation. Your program can issue DISPLAY to obtain the negotiated values for the mode session limit.

No **CNOS** negotiation occurs when setting the limits for a single session (that is, the two LUs do not negotiate drain settings or responsibility values). Therefore, coordinate the mode definition parameter settings between partner LUs using a single-session connection by defining a single session mode at each node.

As part of setting up the initial limits, **CNOS** also sets the guaranteed (that is, the minimum) number of contention-winner and contention-loser sessions and sets the automatic activation count for the source LUs contention-winner sessions. The action of **CNOS** normally affects only the group of sessions with the specified mode name between the source LU and the target LU. Alternatively, one **CNOS** can reset the session limits of all modes for a partner LU.

APPC enforces the new mode session limit and the contention-winner polarities until one side or the other changes them by issuing a subsequent **CNOS** verb. The **CNOS** transaction is invisible at the target LU's API, regardless of which LU is the target. The results of the **CNOS** transaction can be obtained using **DISPLAY**.

Setting a Session Limit to Zero

After **CNOS** raises the session limit above zero, it can reset the limit to zero only. It cannot set the session limit to a value that is not zero, and it cannot redistribute the number of sessions allocated as the contention winners and losers. Therefore, your program cannot change the mode session limits if the two logical units (LUs) have already set the limits to a nonzero value, regardless of which LU initiated the **CNOS** transaction.

A program can change the session limits from a nonzero value, as long as the program first changes the session limit to zero. For example, if the session limit is 8, a program can change it to 6 by first issuing **CNOS** and changing the session limit to zero, and then issuing **CNOS** again and setting the session limit to 6.

APPC can activate one or more LU-LU sessions with the specified mode name as a result of initializing the session limit. You cannot use **CNOS** to activate sessions between two LUs on the same server. APPC deactivates all LU-LU sessions for the specified mode name (or for all mode names for a partner LU) as a result of resetting the session limit to zero. APPC deactivates each session as it becomes free and does not interrupt active conversations.

A separate value, the maximum negotiable session limit, is used in **CNOS** negotiations. If the **set_negotiable** value is AP_YES, the mode session limit value given in this **CNOS** verb also sets the maximum negotiable session limit.

The **lu_alias** and **plu_alias** parameters are 8-byte ASCII character strings. If the name is fewer than eight bytes, it must be padded on the right with ASCII spaces.

You can specify the SNA-defined mode name SNASVCMG for **mode_name**. Use this mode only in a **CNOS** transaction when the source LU and the target LU use parallel user sessions. However, when resetting the session limits to zero for the SNASVCMG PU 2.1 node, the session limits of all other modes between the two LUs must be reset first. The PU 2.1 mode name is a type A EBCDIC character string. A mode name consisting of all spaces is supported. The SNA-defined mode name CPSVCMG is not allowed.

When specifying **plu_mode_sess_lim**, if the mode session limit is currently greater than zero, the value of this parameter must be zero. **CNOS** can raise the limit above zero, but the next **CNOS** must set the value to zero. A single **CNOS** cannot change the mode session limit from one nonzero number to another.

When raising the mode session limit above zero for a parallel-session connection, the target LU can negotiate its parameter to a value greater than zero and less than the specified session limit. The specified or negotiated limit then becomes the new mode session limit and is returned in this field.

The value specified for this parameter must be greater than or equal to the sum of the values specified in the **CNOS min_conwinners_source** and **min_conwinners_target** parameters.

Do not reset the SNASVCMG session limit to zero until all other mode session limits between the two LUs are reset to zero and the count of active sessions for all modes (except SNASVCMG) for the partner LU is zero.

The mode session limit should be large enough to accommodate all active conversations in the mode for all TPs.

For **min_conwinners_source** and **min_conwinners_target**, the sum of both parameters cannot exceed the mode session limit. For single-session connections, these parameters specify the desired contention-winner sessions for the target and source LUs. For the SNASVCMG mode name (with a mode session limit of 2 or 1), the specified minimum number of contention-winner sessions for the target LU must be 1. For the source LU, with a mode session limit of 2, the number must be 1; with a mode session limit of 1, the number must be 0. APPC uses these parameters only when the mode session limit is set to a nonzero value.

APPC uses auto_act only when the mode session limit is set to a nonzero value. If the value is greater than the min_conwinners_source value, APPC uses the new minimum number of contention winners for the source LU as the autoactivation limit.

⚠**Caution**
The **auto_act** parameter can conflict with the on-demand definition of a connection. Autoactivations by either peer partner c an re-establish sessions and connections, possibly resulting in a thrashing situation. Therefore, avoid specifying autoactivatio n between peer PU 2.1 nodes using on-demand connections.

Whether an LU deactivates a session immediately after the current conversation or after all queued conversations are complete depends on the **drain_source** and **drain_target** parameters.

If an LU is to drain its waiting (outbound) allocation requests, it continues to allocate conversations to active sessions. The responsible LU deactivates a session only when the conversation allocated to the session is deallocated and no request is waiting for allocation to any session with the specified mode name between the two LUs. The allocation of a waiting request takes precedence over the deactivation of a session.

If an LU is not to drain its waiting (outbound) allocation requests, the responsible LU deactivates a session as soon as the conversation allocated to the session is deallocated. If no conversation is allocated to the session, the responsible LU deactivates the session immediately. However, this verb does not force deallocation of active conversations.

The **responsible** and **mode_name_select** parameters are interrelated as follows:

- APPC ignores the **responsible** parameter for mode names for which the session limit is currently zero if this **CNOS** verb specifies **mode_name_select** (AP_ALL).

- If CNOS specifies **mode_name_select** (AP_ONE) with a mode session limit of zero, and the current session limit for that mode name is already zero, the **responsible** parameter must specify the same LU (SOURCE or TARGET) that is currently responsible for deactivating sessions. APPC uses this parameter only when **CNOS** specifies a mode session limit of zero.

For parallel-session connections, the **drain_source** and **mode_name_select** parameters are interrelated as follows:

- If **CNOS** specifies **mode_name_select** (AP_ALL) and **drain_source** (AP_YES), APPC ignores **drain_source** for those mode names for which the session limit is currently zero.

- If **CNOS** specifies **mode_name_select** (AP_ALL) and **drain_source** (AP_NO), APPC accepts **drain_source** for all mode names. APPC ends draining for any mode currently draining its requests.

- If **CNOS** specifies **mode_name_select** (AP_ONE), and **drain_source** (AP_YES) is currently in effect, **drain_source** (AP_NO) directs APPC to end the draining at the source LU for requests for the specified mode name.

- If **CNOS** specifies **mode_name_select** (AP_ONE) and **drain_source** (AP_NO) is currently in effect, your program must specify **drain_source** (AP_NO) again.

For parallel-session connections, the **drain_target** parameter and the **mode_name_select** parameter are interrelated as follows:

- If **CNOS** specifies **mode_name_select** (AP_ALL) and **drain_target** (AP_YES), APPC ignores **drain_target** for the mode names for which the session limit is currently zero.

- If **CNOS** specifies **mode_name_select** (AP_ALL) and **drain_target** (AP_NO), APPC accepts **drain_target** for all mode names, regardless of the current session limit. Any draining of waiting (outbound) allocation requests at the target LU is ended.

- If **CNOS** specifies **mode_name_select** (AP_ONE) and **drain_target** (AP_YES) is currently in effect, **drain_target** (AP_NO) ends the target LUs draining.

- If **CNOS** specifies **mode_name_select** (AP_ONE) and **drain_target** (AP_YES), and **drain_target** (AP_NO) is currently in effect, the target LU can either accept **drain_target** (AP_YES) or negotiate the parameter to AP_NO. After the target LU accepts the **drain_target** (AP_YES) parameter, it can drain any remaining waiting (outbound) allocation requests.

DEACTIVATE_SESSION

The **DEACTIVATE_SESSION** verb requests Microsoft® Host Integration Server to deactivate a particular session between the local logical unit (LU) and a specified partner LU, or all sessions on a particular mode.

The following structure describes the verb control block used by the **DEACTIVATE_SESSION** verb.

Syntax

```
typedef struct deactivate_session {
    unsigned short  opcode;
    unsigned char   reserv2[2];
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   reserv3[8];
    unsigned char   lu_alias[8];
    unsigned char   session_id[8];
    unsigned char   plu_alias[8];
    unsigned char   mode_name[8];
    unsigned char   type;
    unsigned char   reserv4[3];
    unsigned short  sense_data;
    unsigned char   fqplu_name[17];
    unsigned char   reserv5[19];
} DEACTIVATE_SESSION;
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_DEACTIVATE_SESSION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

reserv3

A reserved field.

lu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the LU.

session_id

Supplied parameter. Provides the 8-byte identifier of the session to deactivate (returned on the **ACTIVATE_SESSION** verb). If this field is set to 8 binary zeros, Host Integration Server deactivates all sessions for the partner LU and mode.

plu_alias

Supplied parameter. Provides the 8-byte ASCII name used locally for the partner LU. If the default remote LU is to be used, fill this parameter with spaces. If the partner LU is to be specified with the **fqplu_name** parameter, fill this parameter with binary zeros.

mode_name

Supplied parameter. Specifies the EBCDIC (type A) mode name.

type

Supplied parameter. Specifies the type of deactivation. Possible values are:

AP_DEACT_CLEANUP

Deactivate the session immediately, without waiting for sessions to end.

AP_DEACT_NORMAL

Do not deactivate the session until all conversations using the session have ended.

sense_data

Returned parameter. Specifies the deactivation sense data for the session.

reserv4

A reserved field.

fqplu_name

Supplied parameter. Provides the partner LU name in EBCDIC (type A) when no **plu_alias** name is defined at the local node and the partner LU is located at a different node. This parameter is ignored if **plu_alias** is specified.

reserv5

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully. The secondary return code indicates the polarity of the established session. The following values can be returned.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error, specified by one of the following secondary return codes:

AP_INVALID_LU_ALIAS

Secondary return code; APPC cannot find the specified **lu_alias** among those defined.

AP_INVALID_PLU_ALIAS

Secondary return code; APPC did not recognize the specified **plu_alias**.

AP_INVALID_SESSION_ID

Secondary return code; APPC did not recognize the specified **session_id**.

AP_INVALID_MODE_NAME

Secondary return code; APPC did not recognize the specified **mode_name**.

AP_INVALID_FQPLU_NAME

Secondary return code; APPC did not recognize the specified **fqplu_name**.

AP_INVALID_TYPE

Secondary return code; APPC did not recognize the specified **type**.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions occurred:

The node used by this conversation encountered an ABEND.

The connection between the TP and the PU 2.1 node has been broken (a local area network error occurred).

The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus,

communication could not take place. Contact the system administrator for corrective action.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

DISPLAY

The **DISPLAY** verb returns configuration information and current operating values for the SNA node.

It is recommended that you use the GetAppcConfig Windows extension function to obtain system configuration information relating to APPC LUs. Users of 5250 emulators, in particular, should use the **GetAPPCConfig** Windows extension.

Note

Because of the nature of client/server architecture, the implementation of the **DISPLAY** verb on Host Integration Server 2009 contains important differences from the IBM Extended Services for OS/2 version 1.0 (IBM ES for OS/2 version 1.0) on which it was based.

Note

For applications that use the APPC **DISPLAY** verb in IBM ES for OS/2 version 1.0 compatibility mode and that do not use the Host Integration Server extensions for enumerating all active servers and connections, Host Integration Server will randomly choose a default **DISPLAY** connection, unless a specific default **DISPLAY** connection has been configured in SNA Manager. This connection is used as the basis for all **DISPLAY** requests. For information about specifying the default **DISPLAY** connection, see Host Integration Server 2009 Help.

The following structure describes the verb control block used by the **DISPLAY** verb.

Syntax

```
struct display {
    unsigned short  opcode;
    unsigned char   reserv2[2];
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned long   init_sect_len;
    unsigned long   buffer_len;
    unsigned char FAR * buffer_ptr;
    unsigned long  num_sections;
    unsigned long  display_len;
    unsigned long  area_needed;
    unsigned char  sna_global_info;
    unsigned char  lu62_info;
    unsigned char  am_info;
    unsigned char  tp_info;
    unsigned char  sess_info;
    unsigned char  link_info;
    unsigned char  lu_0_3_info;
    unsigned char  gw_info;
    unsigned char  x25_physical_link_info;
    unsigned char  sys_def_info;
    unsigned char  adapter_info;
    unsigned char  lu_def_info;
    unsigned char  plu_def_info;
    unsigned char  mode_def_info;
    unsigned char  link_def_info;
    unsigned char  ms_info;
    struct sna_global_info_sect FAR * sna_global_info_ptr;
    struct lu62_info_sect FAR * lu62_info_ptr;
    struct am_info_sect FAR * am_info_ptr;
    struct tp_info_sect FAR * tp_info_ptr;
    struct sess_info_sect FAR * sess_info_ptr;
    struct link_info_sect FAR * link_info_ptr;
    struct lu_0_3_info_sect FAR * lu_0_3_info_ptr;
    struct gw_info_sect FAR * gw_info_ptr;
    struct x25_physical_link_info_sect FAR * x25_physical_link_info_ptr;
    struct sys_def_info_sect FAR * sys_def_info_ptr;
    struct adapter_info_sect FAR * adapter_info_ptr;
    struct lu_def_info_sect FAR * lu_def_info_ptr;
```

```
    struct plu_def_info_sect FAR * plu_def_info_ptr;
    struct mode_def_info_sect FAR * mode_def_info_ptr;
    struct link_def_info_sect FAR * link_def_info_ptr;
    struct ms_info_sect FAR * ms_info_ptr;
} DISPLAY;
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_DISPLAY.

reserv2

A reserved field, this value must be set to NULL.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

init_sect_len

Supplied parameter. Specifies the number of bytes in the initial section of the VCB, up to the beginning of information pointers. This parameter and the **num_sections** parameter must be set to specific values depending on the format being requested. See the notes below for details.

buffer_len

Supplied parameter. Specifies the length (0 to 65535 bytes) of the passed display data buffer.

buffer_ptr

Supplied parameter. Provides the address of the display data buffer that will contain the requested information.

num_sections

Supplied parameter. Specifies the maximum number of information sections that can be returned by the verb. This parameter and the **init_sect_len** parameter must be set to specific values depending on the format being requested. See the notes below for details.

display_len

Returned parameter. Provides the total number of bytes used that are returned into the display data buffer.

area_needed

Returned parameter. Provides the total number of bytes needed for all of the displayed data.

sna_global_info

Supplied parameter. Specifies if global information is requested. Allowed values are AP_YES and AP_NO.

lu62_info

Supplied parameter. Specifies if information on all active LUs, their partners, and their modes is requested. Allowed values are AP_YES and AP_NO.

am_info

Supplied parameter. Specifies if Attach Manager information on the defined TP is requested. Allowed values are AP_YES and AP_NO.

Note
This option is not supported by Host Integration Server and this parameter must be set to AP_NO.

tp_info

Supplied parameter. Specifies if information on the active TPs and any active conversations is requested. Allowed values are AP_YES and AP_NO.

Note
This option is not supported by Host Integration Server and this parameter must be set to AP_NO.

sess_info

Supplied parameter. Specifies if information on sessions is requested. Allowed values are AP_YES and AP_NO.

link_info

Supplied parameter. Specifies if information on the active SNA logical lines is requested. Allowed values are AP_YES and AP_NO.

lu_0_3_info

Supplied parameter. Specifies if information on logical units type 0, 1, 2, and 3 is requested. Allowed values are AP_YES and AP_NO.

gw_info

Supplied parameter. Specifies if information on the SNA gateway is requested. Allowed values are AP_YES and AP_NO.

x25_physical_link_info

Supplied parameter. Specifies if X.25 information is required. Allowed values are AP_YES and AP_NO.

Note
This option is not supported by Host Integration Server and this parameter must be set to AP_NO.

sys_def_info

Supplied parameter. Specifies if information about the default LU, node names, and default parameters for inbound and outbound implicit partners is requested. Allowed values are AP_YES and AP_NO.

adapter_info

Supplied parameter. Specifies if information about the configured communications adapters is requested. Allowed values are AP_YES and AP_NO. This parameter must be set to AP_NO when NS/2 format is requested.

lu_def_info

Supplied parameter. Specifies if information about the defined LUs is requested. Allowed values are AP_YES and AP_NO.

plu_def_info

Supplied parameter. Specifies if information about the defined partner LUs is requested. Allowed values are AP_YES and AP_NO.

mode_def_info

Supplied parameter. Specifies if information about the defined nodes is requested. Allowed values are AP_YES and AP_NO.

link_def_info

Supplied parameter. Specifies if information about the defined logical links is requested. Allowed values are AP_YES and AP_NO.

ms_info

Supplied parameter. Specifies if information about management services is requested. Allowed values are AP_YES and AP_NO. This parameter must be set to AP_NO when NS/2 format is requested.

sna_global_info_ptr

Returned parameter. Indicates the address of the beginning of SNA global information in the data buffer.

lu62_info_ptr

Returned parameter. Indicates the address of the beginning of LU 6.2 information in the data buffer.

am_info_ptr

Returned parameter. Indicates the address of the beginning of the Attach Manager information in the data buffer.

Note
This option is not supported by Host Integration Server.

tp_info_ptr

Returned parameter. Indicates the address of the beginning of TP information in the data buffer.

Note
This option is not supported by Host Integration Server.

sess_info_ptr

Returned parameter. Indicates the address of the beginning of session information in the data buffer.

link_info_ptr

Returned parameter. Indicates the address of the beginning of link information in the data buffer.

lu_0_3_info_ptr

Returned parameter. Indicates the address of the beginning of LU information in the data buffer.

gw_info_ptr

Returned parameter. Indicates the address of the beginning of gateway information in the data buffer.

x25_physical_link_info_ptr

Returned parameter. Indicates the address of the beginning of X.25 information in the data buffer.

Note
This option is not supported by Host Integration Server.

sys_def_info_ptr

Returned parameter. Indicates the address of the beginning of system default information in the data buffer.

adapter_info_ptr

Returned parameter. Indicates the address of the beginning of adapter information in the data buffer.

lu_def_info_ptr

Returned parameter. Indicates the address of the beginning of local LU definition information in the data buffer.

plu_def_info_ptr

Returned parameter. Indicates the address of the beginning of partner LU definition information in the data buffer.

mode_def_info_ptr

Returned parameter. Indicates the address of the beginning of mode definition information in the data buffer.

link_def_info_ptr

Returned parameter. Indicates the address of the beginning of link definition information in the data buffer.

ms_info_ptr

Returned parameter. Indicates the address of the beginning of management services information in the data buffer.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_DISPLAY_INVALID_CONSTANT

Secondary return code; the value supplied for NUM_SECTIONS or INIT_SEC_LEN is invalid.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_DISPLAY_INFO_EXCEEDS_LEN

Secondary return code; the returned **DISPLAY** information did not fit in the buffer.

AP_INVALID_DATA_SEGMENT

Secondary return code; the segment containing the data buffer is too small for the specified data length.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation has encountered an ABEND.

The connection between the TP and the node type 2.1 has been broken (a LAN error).

The SnaBase at the TPs computer has encountered an ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

DISPLAY identifies an LU by alias alone. If the same local LU alias is used multiple times in a domain (for backup or other purposes) and that LU alias is specified through **DISPLAY**, the verb can flow to a different LU than the one intended.

For the **DISPLAY** verb to return successfully, a specific connection must be defined in the SNA Manager program **Display Verb** dialog box. IBM originally defined the **DISPLAY** verb with the IBM OS/2 Extended Edition product which assumed a single connection. However, because Host Integration Server supports multiple connections, the specific connection associated with the **DISPLAY** verb must be configured.

The **DISPLAY** verb requires a user-supplied buffer for the return of system information. If the buffer is not large enough, APPC returns the AP_DISPLAY_INFO_EXCEEDS_LEN return code, along with the size actually needed at the time of the request (in the **area_needed** parameter). One possible strategy for the use of this verb follows:

- If the **buffer_len** value is less than the **area_needed** value returned by APPC, and the required length is less than 64 kilobytes (KB), then increase the size of the display buffer to equal or greater than the **area_needed** value.

- If the **area_needed** value is greater than 64KB, you can choose to request each information section individually. Or, you can take the following steps:

    1. Process the information sections with complete information, whose total number displayed equals the total actual number.

2. Choose a subset of the information sections you requested that contains incomplete information, and reissue the verb requesting those information sections.

3. Repeat steps a and b as needed.

Note
If an individual information section is greater than 64 KB, then you cannot get all of the requested information from AP PC.

The **DISPLAY** verb should not be executed from different threads of the same process, since it is not thread-safe.

The **DISPLAY** verb returns AP_DISPLAY_INVALID_CONSTANT if the following values are not set for the supplied parameters for **init_sect_len** and **num_sections**:

	NS/2 format	**IBM EE format**	**NS/2 format (Windows 2000 only)**	**IBM EE format (Windows 2000 only)**
init_sect_len	50	44	52	48
num_sections	16	9	16	9

The AP_DISPLAY_INVALID_CONSTANT is also returned when the following parameters are not set properly:

- **reserv2** must be set to NULL.
- **am_info** must be set to AP_NO.
- **tp_info** must be set to AP_NO.
- **adapter_info** must be set to AP_NO if NS/2 format is requested.
- **ms_info** must be set to AP_NO if NS/2 format is requested.

See Also
Reference
Host Integration Server Extensions
Other Resources
Differences by Information Type

Host Integration Server Extensions

The Host Integration Server **DISPLAY** verb is compatible with the IBM ES for OS/2 version 1.0 **DISPLAY** verb. However, since IBM ES for OS/2 version 1.0 is a single-server system and Host Integration Server supports multiple-server systems, the **DISPLAY** verb has been extended to allow the user to target a specific server running Host Integration Server by which the **DISPLAY** verb will be processed.

To direct a **DISPLAY** verb at a particular server running Host Integration Server, place the ASCII string CSEXTNID, followed by the computer name of the server running Host Integration Server at the start of the buffer pointed to by **buffer_ptr**. The computer name is a 32-byte ASCII string and can be zero or padded with spaces.

Because the local node identifier is configured on a per-node basis for IBM ES for OS/2 version 1.0 and can be different for each connection in Host Integration Server, Host Integration Server also allows you to specify an optional connection name. This is an 8-byte ASCII string, which is placed after the 32-byte computer name. Again, the string can be zero or padded with spaces. The following example illustrates the CSEXTNID extension:

csextnid computername 0000000000000000000 name

If you do not specify a connection name, Host Integration Server returns information about the first connection configured for the Host Integration Server system.

If you do not specify a computer name, Host Integration Server will randomly choose a default **DISPLAY** computer and connection, unless a specific default **DISPLAY** connection has been configured on the server. These parameters can be configured with the SNA Manager or the Host Integration Server Administrator Client when using Host Integration Server 2009. **DISPLAY** will behave as if you specified the connection and the computer name of the server that owns the verb. For additional information about using default LUs, see Host Integration Server 2009 Help.

Host Integration Server also allows you to use **DISPLAY** to return a list of active servers. To do so, place the string CSEXTNIDCSLISTND in the **DISPLAY** buffer and set the supplied parameters **sna_global_info**, **lu62_info**, and so on, to AP_NO. The information is returned in the **DISPLAY** buffer in the following format.

Syntax

```
#activenodes            - 2 bytes
  node_name 1           - 8 bytes
  box_name 1            - 32 bytes
  .
  node_name m
  box_name m
```

Remarks

In the current version of Host Integration Server, **node_name** is always SNASERVR and **box_name** is the computer name of the server.

Differences by Information Type

Differences in the implementation of the **DISPLAY** verb between Host Integration Server 2009 and IBM ES for OS/2 are described in this section by information type. For each information type, there is a topic that describes:

- The information defined by IBM ES for OS/2 version 1.0.
- The information returned by Host Integration Server.

Note
Host Integration Server does not support all of the information types supported by IBM ES for OS/2 version 1.0. If an information type is not listed in this section, it is not supported by Host Integration Server.

In This Section

SNA Global Information

SNA global information is defined or returned as described here.

Defined by IBM ES for OS/2 version 1.0

Information on SNA global information is provided in the **sna_global_info_sect** structure as defined below.

```
typedef struct sna_global_info_sect {
    unsigned char version;
    unsigned char release;
    unsigned char net_name[8];
    unsigned char pu_name[8];
    unsigned char node_id[4];
    type_product_set_id product_set_id;
    unsigned char alias_cp_name[8];
    unsigned char node_type;
    unsigned char cp_nau_addr;
    unsigned char corr_serv_disk;
    unsigned char reserved;
    unsigned char appc_version;
    unsigned char appc_release;
    unsigned char appc_fixlevel;
} SNA_GLOBAL_INFO_SECT;
```

Members

version

Communications Manager Extended Edition version number.

release

Communications Manager Extended Edition release number.

net_name

Network name, first part of fully qualified control program (CP) name, in EBCDIC (type A).

pu_name

PU name, second part of fully qualified CP name, in EBCDIC (type A).

node_id

4-byte hexadecimal exchange identifier.

product_set_id

Computer product data.

alias_cp_name

Node name (local name for CP) in ASCII.

node_type

AP_NN, AP_EN, or AP_LEN.

cp_nau_addr

CP NAU address where 0 means not used (an independent LU). Other legal values are 1 to 254.

corr_serv_disk

Last four digits of corrective service disk number.

reserved

Reserved field.

appc_version

APPC version number.

appc_release

APPC release number.

appc_fixlevel

APPC patch number.

Returned by Host Integration Server

Information on SNA global information is provided in the **sna_global_info_sect** structure defined below.

```
typedef struct sna_global_info_sect {
    unsigned char version;
    unsigned char release;
    unsigned char net_name[8];
    unsigned char pu_name[8];
    unsigned char node_id[4];
    type_product_set_id product_set_id;
    unsigned char alias_cp_name[8];
    unsigned char node_type;
    unsigned char cp_nau_addr;
    unsigned char corr_serv_disk;
    unsigned char reserved;
    unsigned char appc_version;
    unsigned char appc_release;
    unsigned char appc_fixlevel;
} SNA_GLOBAL_INFO_SECT;
```

Members

version

Major operating system (OS) version number.

release

Minor OS version number.

net_name

Node network name in EBCDIC (type A).

pu_name

PU name in EBCDIC (type A) associated with connection.

node_id

Node identifier to send.

product_set_id

Set to EBCDIC zeros.

alias_cp_name

Node name, local name for the control program (CP), in ASCII.

node_type

Set to AP_LEN.

cp_nau_addr

CP NAU address where 0 means not used (an independent LU). Other legal values are 1 to 254.

corr_serv_disk

Reserved field set to zero.

reserved

Reserved field set to zero.

appc_version

Host Integration Server major version number.

appc_release

Host Integration Server minor version number.

appc_fixlevel

Host Integration Server patch number.

Remarks

Host Integration Server returns **version** and **release** as the major and minor OS version numbers from **GetVersion**. Because Host Integration Server 2009 has no information on the computer type, serial number, and manufacturer, **product_set_id** is set to EBCDIC zeros.

Host Integration Server does not support APPN node types, so the node type is returned as 1 (an AP_LEN node), and not 2 or 3 (AP_NN or AP_EN nodes), as defined by IBM ES for OS/2 version 1.0.

LU 6.2 Information

Information on LUs is provided in the **lu62_info_sect** structure as defined below.

Syntax

```
typedef struct lu62_info_sect {
    unsigned long  lu62_init_sect_len;
    unsigned short num_lu62s;
    unsigned short total_lu62s;
} LU62_INFO_SECT;
```

Members

lu62_init_sect_len

Structure length.

num_lu62s

Number of configured LUs displayed.

total_lu62s

Total number of configured LUs.

For each configured LU, an **lu62_overlay** structure is provided as defined below.

Syntax

```
typedef struct lu62_overlay {
    unsigned long  lu62_entry_len;
    unsigned long  lu62_overlay_len;
    unsigned char  lu_name[8];
    unsigned char  lu_alias[8];
    unsigned short num_plus;
    unsigned char  fqlu_name[17];
    unsigned char  default_lu;
    unsigned char  reserv3;
    unsigned char  lu_local_addr;
    unsigned short lu_sess_lim;
    unsigned char  max_tps;
    unsigned char  lu_type;
} LU62_OVERLAY;
```

Members

lu62_entry_len

Size of this LU entry.

lu62_overlay_len

This value contains **sizeof(**struct **lu62_overlay)**–**sizeof(lu62_entry_len)**.

lu_name

LU name (EBCDIC type A).

lu_alias

LU alias (ASCII).

num_plus

Number of partner LUs.

fqlu_name

Fully qualified LU name (EBCDIC type A).

default_lu

For local LU group, an LU equal to the **default_lu** is used if none is specified. Legal values are AP_NO and AP_YES.

On Host Integration Server, there is no concept of a default local LU. Therefore, the **default_lu** flag, which is set to AP_YES for the node in IBM ES for OS/2 version 1.0, is set to AP_NO for Host Integration Server.

lu_local_addr

NAU address, 0–254.

lu_sess_lim

Configured session limit, 0–255.

max_tps

Maximum number of TPs, 1–255.

lu_type

Always LU type 6.2.

For each configured LU, a **plu_62_overlay** structure for the partner LU is provided as defined below.

Syntax

```
typedef struct plu62_overlay {
    unsigned long  plu62_entry_len;
    unsigned long  plu62_overlay_len;
    unsigned char  plu_alias[8];
    unsigned short num_modes;
    unsigned char  plu_un_name[8];
    unsigned char  fqplu_name[17];
    unsigned char  reserv3;
    unsigned char  plu_sess_lim;
    unsigned char  dlc_name[8];
    unsigned char  adapter_num;
    unsigned char  dest_addr_len;
    unsigned char  dest_addr[32];
    unsigned int   par_sess_supp:1;
    unsigned int   reserv4:7;
    unsigned int   def_already_ver:1;
    unsigned int   def_conv_sec:1;
    unsigned int   def_sess_sec:1;
    unsigned int   reserv5:5;
    unsigned int   act_already_ver:1;
    unsigned int   act_conv_sec:1;
    unsigned int   reserv6:6;
    unsigned int   implicit_part:1;
    unsigned int   reserv7:7;
} PLU62_OVERLAY;
```

Members

plu62_entry_len

Size of this partner LU entry.

plu62_overlay_len

This value contains **sizeof(**struct **plu62_overlay)**–**sizeof(plu62_entry_len)**.

plu_alias

Partner LU alias (ASCII).

num_modes

Number of modes.

plu_un_name

Partner LU uninterpreted name (EBCDIC).

fqplu_name

Fully qualified partner LU name (EBCDIC type A).

reserv3

Reserved field set to zero.

plu_sess_lim

Partner LU session limit, 0–255.

dlc_name

DLC name (ASCII).

adapter_num

DLC adapter number.

dest_addr_len

Length of destination adapter address.

dest_addr

Destination adapter address.

par_sess_supp

Bit 15 of a bitfield specifying parallel sessions. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

reserv4

Bits 8–14 of a bitfield specifying a reserved field set to zero.

def_already_ver

Bit 7 of a bitfield specifying whether the configured already verified option is supported. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

def_conv_sec

Bit 6 of a bitfield specifying whether the configured conversation security option is supported. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

def_sess_sec

Bit 5 of a bitfield specifying whether the configured session security option is supported. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

reserv5

Bits 0–4 of a bitfield specifying a reserved field set to zero.

act_already_ver

Bit 15 of a bitfield specifying whether the active already verified option is supported. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

act_conv_sec

Bit 14 of a bitfield specifying whether the active conversation security option is supported. Valid values are AP_NOT_SUPPORTED and AP_SUPPORTED.

reserv6

Bits 8–13 of a bitfield specifying a reserved field set to zero.

implicit_part

Bit 7 of a bitfield specifying whether this is an implicit partner. Valid values are AP_NO and AP_YES.

For partner LU group, **implicit_part** indicates the partner LU group was configured as an implicit primary logical unit (PLU).

reserv7

Bits 0–6 of a bitfield specifying a reserved field set to zero.

Remarks

Host Integration Server returns information on all the configured LU 6.2s in the system, including the implicit PLU and all instances of implicit modes. IBM ES for OS/2 version 1.0 only returns information on those that are in use or have been in use.

For partner LU group, **implicit_part** indicates the partner LU group was configured as an implicit primary logical unit (PLU).

For mode group, **implicit_mode** bitfield returned in the **mode_overlay** structure indicates the mode group was configured as an implicit mode.

Session Information

Information on session information is provided in the **sess_info_sect** structure as defined below.

Syntax

```
typedef struct sess_info_sect {
    unsigned long  sess_sect_len;
    unsigned short num_sessions;
    unsigned short total_sessions;
} SESS_INFO_SECT;
```

Members

sess_sect_len

The length of the initial session information section, including this parameter, up to the first session group. The length does not include any previous information sections.

num_sessions

The number of session groups returned by the **DISPLAY** verb into your program's buffer. This is the number of times the session group is repeated.

total_sessions

The total number of session groups. This number is the same as the number returned in the **num_sessions** member except when APPC has more information about session groups than it can place in the supplied buffer, in which case this number is larger.

For each session group, a **sess_overlay** structure for the session is provided as defined below.

Syntax

```
typedef struct sess_overlay {
    unsigned long  sess_entry_len;
    unsigned long  reserv3;
    unsigned char  sess_id[8];
    unsigned long  conv_id[8];
    unsigned char  lu_alias[8];
    unsigned char  plu_alias[8];
    unsigned char  mode_name[8];
    unsigned short send_ru_size;
    unsigned short rcv_ru_size;
    unsigned short send_pacing_size;
    unsigned short rcv_pacing_size;
    unsigned char  link_id[12];
    unsigned char  daf;
    unsigned char  oaf;
    unsigned char  odai;
    unsigned char  sess_type;
    unsigned char  conn_type;
    unsigned char  reserv4;
    FPCID_OVERLAY  fpcid;
    unsigned char  cgid[4];
    unsigned char  fqlu_name[17];
    unsigned char  fqplu_name[17];
    unsigned char  pacing_type;
    unsigned char  reserv5;
 } SESS_OVERLAY;
```

Defined by IBM ES for OS/2 version 1.0

Members

sess_entry_len

Size of this session group entry.

sess_id

The internal identifier of the session for which this information is displayed.

conv_id

The unique four-byte ID of the conversation currently using this session.

lu_alias

LU alias (ASCII).

plu_alias

Partner LU alias (ASCII).

mode_name

The name of the mode (EBCDIC).

send_ru_size

The maximum RU size used on this session and this **mode_name** for sending RUs.

rcv_ru_size

The maximum RU size used on this session and this **mode_name** for receiving RUs.

send_pacing_size

The size of the send pacing window on this session.

rcv_pacing_size

The size of the receive pacing window on this session.

link_id

Name of local logical link station.

daf

The destination address field for this session.

oaf

The origin address field for this session.

odai

The origin destination address indicator field for this session.

sess_type

The type of the session. The session type can be one of the following:

SSCP_PU_SESSION

This session is between a workstation physical unit and a host system services control point. This type of session exists if the local node contains a dependent LU, or if the session has been solicited in order to send alerts to the host.

SSCP_LU_SESSION

This session is between a dependent LU and a host system services control point.

LU_LU _SESSION

This session is between two LUs.

conn_type

Indicates whether the session activation protocol follows the rules for an independent LU or a dependent LU. The connection type can be one of the following:

AP_HOST_SESSION

For dependent LU protocols, the workstation LU is defined as dependent at the host, the host LU sends the session activation request (BIND), and each workstation LU can support only one session at a time.

AP_PEER_SESSION

For independent LU protocols, an LU can send a BIND, and can have multiple sessions to different partners, or parallel sessions to the same partner LU.

fq_pc_id

Fully qualified procedure correlation identifier of the session.

cgid

Unique identifier for the conversation group of the session.

fqlu_name

The fully-qualified LU name in EBCDIC (type A).

fqplu_name

The fully-qualified partner LU name in EBCDIC (type A).

pacing_type

The pacing type can be one of the following:

AP_FIXED

Fixed pacing.

AP_ADAPTIVE

Adaptive pacing.

Returned by Host Integration Server

Members
sess_entry_len

Size of this session group entry.

sess_id

The internal identifier of the session for which this information is displayed.

conv_id

The unique four-byte ID of the conversation currently using this session.

lu_alias

LU alias (ASCII).

plu_alias

Partner LU alias (ASCII).

mode_name

The name of the mode (EBCDIC).

mode_name

The name of the mode (EBCDIC).

send_ru_size

The maximum RU size used on this session and this **mode_name** for sending RUs.

rcv_ru_size

The maximum RU size used on this session and this **mode_name** for receiving RUs.

send_pacing_size

The size of the send pacing window on this session.

rcv_pacing_size

The size of the receive pacing window on this session.

link_id

Connection name.

daf

The destination address field for this session.

oaf

The origin address field for this session.

odai

The origin destination address indicator field for this session.

sess_type

The type of the session. The session type can be one of the following:

SSCP_PU_SESSION

This session is between a workstation physical unit and a host system services control point. This value is never returned by Host Integration Server.

SSCP_LU_SESSION

This session is between a dependent LU and a host system services control point.

LU_LU _SESSION

This session is between two LUs.

conn_type

Indicates whether the session activation protocol follows the rules for an independent LU or a dependent LU. The connection type can be one of the following:

AP_HOST_SESSION

For dependent LU protocols, the workstation LU is defined as dependent at the host, the host LU sends the session activation request (BIND), and each workstation LU can support only one session at a time.

AP_PEER_SESSION

For independent LU protocols, an LU can send a BIND, and can have multiple sessions to different partners, or parallel sessions to the same partner LU.

AP_BOTH_SESSION

Connections can support both Dependent and Independent LUs.

fq_pc_id

Set to zero.

cgid

Set to zero.

type_of_pacing

The pacing type can be one of the following:

AP_FIXED

Fixed pacing.

AP_ADAPTIVE

Adaptive pacing. This value is never returned by Host Integration Server.

Active Link Information

Active link information is provided in the **link_info_sect** structure as defined below.

Syntax

```
typedef struct link_info_sect {
    unsigned long  link_init_sect_len;
    unsigned short num_links;
    unsigned short total_links;
} LINK_INFO_SECT;
```

Members

link_init_sect_len

The length of the initial active link information section, including this parameter, up to the first link overlay group. The length does not include any previous information sections.

num_links

The number of active links returned by the **DISPLAY** verb into your program's buffer. This is the number of times the link overlay group is repeated.

total_links

The total number of active links. This number is the same as the number returned in the **num_links** member except when APPC has more information about active links than it can place in the supplied buffer, in which case this number is larger.

For each active link, a **link_overlay** structure for the active link is provided as defined below.

Syntax

```
typedef struct link_overlay {
    unsigned long  link_entry_len;
    unsigned char  link_id[12];
    unsigned long  dlc_name[8];
    unsigned char  adapter_num;
    unsigned char  dest_addr_len;
    unsigned char  dest_addr[32];
    unsigned char  inbound_outbound;
    unsigned char  state;
    unsigned char  deact_link_flag;
    unsigned char  reserv3;
    unsigned short num_sessions;
    unsigned short ru_size;
    unsigned short reserv4;
    unsigned char  adj_fq_cp_name[17];
    unsigned char  adj_node_type;
    unsigned char  reserv5;
    unsigned char  cp_cp_sess_spt;
    unsigned char  conn_type;
    unsigned char  ls_role;
    unsigned char  line_type;
    unsigned char  tg_number;
    unsigned long  eff_capacity;
    unsigned char  conn_cost;
    unsigned char  byte_cost;
    unsigned char  propagation_delay;
    unsigned char  user_def_1;
    unsigned char  user_def_2;
    unsigned char  user_def_3;
    unsigned char  security;
    unsigned char  reserv6;
 } LINK_OVERLAY;
```

Defined by IBM ES for OS/2 version 1.0

Members

link_entry_len

Size of this link entry.

link_id

Local logical link station name (EBCDIC).

dlc_name

Data link control (DLC) name set to one of the following:

ETHERAND

IBMTRNET

IBMPCNET

SDLC

TWINAX (Not supported by Host Integration Server 2009)

X25DLC

adapter_num

Adapter number used by this link to connect to the adjacent node.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

inbound_outbound

The direction of the link. Values can be:

AP_OUTBOUND

The link is outbound.

AP_INBOUND

The link is inbound.

state

The state of the link. The link state can be one of the following:

AP_CONALS_PND

The process to bring up the link has started but XID negotiation has not started.

AP_XID_PND

XID negotiation is in process.

AP_CONTACT_PND

XID negotiation has been completed but the final response from the DLC has not been received.

AP_CONTACTED

The link is fully functioning.

AP_DISC_PND

A request to disconnect the link has been issued to the DLC.

AP_DISC_RQ

The operator has requested that the link be disconnected.

deact_link_flag

Deactivate logical link.

reserv3

A reserved field.

num_sessions

Number of active sessions.

ru_size

RU size.

reserv4

A reserved field.

adj_fq_cp_name

Fully qualified **cp_name** in adjacent node.

adj_node_type

The adjacent node type (NN, EN, or LEN).

cp_cp_sess_spt

Specifies whether the link supports CP-CP sessions.

conn_type

Indicates whether the session activation protocol follows the rules for an independent LU or a dependent LU. The connection type can be one of the following:

AP_HOST_SESSION

For dependent LU protocols, the workstation LU is defined as dependent at the host, the host LU sends the session activation request (BIND), and each workstation LU can support only one session at a time.

AP_PEER_SESSION

For independent LU protocols, an LU can send a BIND, and can have multiple sessions to different partners, or parallel sessions to the same partner LU.

AP_BOTH_SESSION

Connections can support both Dependent and Independent LUs.

ls_role

Specifies the link station role.

line_type

The line type.

tg_number

Transmission group number.

eff_capacity

Highest bit rate transmission effective capacity supported.

conn_cost

Relative cost per connection time using this link.

byte_cost

Relative cost of transmitting a byte over link.

propagation_delay

Indicates amount of time for signal to travel length of link. Set to one of the following:

AP_PROP_DELAY_MINIMUM

AP_PROP_DELAY_LAN

AP_PROP_DELAY_TELEPHONE

AP_PROP_DELAY_PKT_SWITCHED_NET

AP_PROP_DELAY_SATELLITE

AP_PROP_DELAY_MAXIMUM

user_def_1

User-defined TG characteristics.

user_def_2

User-defined TG characteristics.

user_def_3

User-defined TG characteristics.

security

The security value for this link. Set to one of the following:

AP_SEC_NONSECURE

AP_SEC_PUBLIC_SWITCHED_NETWORK

AP_SEC_UNDERGROUND_CABLE

AP_SEC_SECURE_CONDUIT

AP_SEC_GUARDED_CONDUIT

AP_SEC_ENCRYPTED

AP_SEC_GUARDED_RADIATION

reserv6

A reserved field.

Returned by Host Integration Server

Members

link_entry_len

Size of this link entry.

link_id

Connection name.

dlc_name

DLC name set to one of the following:

IBMTRNET

SDLC

X25DLC

adapter_num

Adapter number used by this link to connect to the adjacent node. Always set to zero.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

inbound_outbound

The direction of the link. Values can be:

AP_OUTBOUND

The link is outbound.

AP_INBOUND

The link is inbound.

state

The state of the link. The link state can be one of the following:

AP_CONALS_PND

The process to bring up the link has started but XID negotiation has not started.

AP_XID_PND

XID negotiation is in process.

AP_CONTACT_PND

XID negotiation has been completed but the final response from the DLC has not been received.

AP_CONTACTED

The link is fully functioning.

AP_DISC_PND

A request to disconnect the link has been issued to the DLC.

AP_DISC_RQ

The operator has requested that the link be disconnected.

deact_link_flag

Deactivate logical link.

num_sessions

Number of active sessions.

ru_size

RU size.

adj_fq_cp_name

Fully qualified **cp_name** in adjacent node. Always set to EBCDIC spaces.

adj_node_type

The adjacent node type. Always set to AP_LEN.

cp_cp_sess_spt

Specifies whether the link supports CP-CP sessions. Always set to AP_NO.

conn_type

Indicates whether the session activation protocol follows the rules for an independent LU or a dependent LU. The connection type can be one of the following:

AP_HOST_SESSION

For dependent LU protocols, the workstation LU is defined as dependent at the host, the host LU sends the session activation request (BIND), and each workstation LU can support only one session at a time.

AP_PEER_SESSION

For independent LU protocols, an LU can send a BIND, and can have multiple sessions to different partners, or parallel sessions to the same partner LU.

ls_role

Specifies the link station role.

line_type

The line type.

tg_number

Transmission group number. Always set to zero.

effective_capacity

Highest bit rate transmission effective capacity supported. Always set to zero.

conn_cost

Relative cost per connection time using this link. Always set to zero.

byte_cost

Relative cost of transmitting a byte over link. Always set to zero.

propagation_delay

Indicates amount of time for signal to travel length of link. This parameter is always set to AP_PROP_DELAY_MAXIMUM.

user_def_1

User-defined TG characteristics. Always set to zero.

user_def_2

User-defined TG characteristics. Always set to zero.

user_def_3

User-defined TG characteristics. Always set to zero.

security

The security value for this link. Always set to AP_SEC_NONSECURE.

LU 0 to 3 Information

LU 0 to 3 information is provided in the **lu_0_3_info_sect** structure as defined below.

Syntax

```
typedef struct lu_0_3_info_sect {
    unsigned long  lu_0_3_init_sect_len;
    unsigned short num_lu_0_3s;
} LU_0_3_ INFO_SECT;
```

Members

lu_0_3_init_sect_len

The length of the initial LU 0 to 3 information section, including this parameter, up to the first link overlay group. The length does not include any previous information sections.

num_lu_0_3s

The number of LU groups. This is the number of times the lu_0_3 overlay group is repeated.

For each configured LU, an **lu_0_3_overlay** structure for the LU is provided as defined below.

Syntax

```
typedef struct lu_0_3_overlay {
    unsigned long  lu_0_3_entry_len;
    unsigned char  access_type;
    unsigned char  lu_type;
    unsigned char  lu_daf;
    unsigned char  lu_short_name;
    unsigned char  lu_long_name[8];
    unsigned char  session_id[8];
    unsigned long  dlc_name[8];
    unsigned char  adapter_num;
    unsigned char  dest_addr_len;
    unsigned char  dest_addr[32];
    unsigned char  sscp_lu_sess_state;
    unsigned char  lu_lu_sess_state;
    unsigned char  link_id[12];
 } LU_0_3_OVERLAY;
```

Defined by IBM ES for OS/2 version 1.0

Members

lu_0_3_entry_len

Size of this LU entry.

access_type

The access type (AP_3270 or AP_LUA).

lu_type

The LU type (AP_LU0, AP_LU1, AP_LU2, or AP_LU3).

lu_daf

The network addressable unit of the LU for which the information is displayed.

lu_short_name

The 1-byte LU short name (ASCII).

lu_long_name

The 8-byte ASCII LU long name.

session_id

The LU-LU session ID.

dlc_name

DLC name set to one of the following:

ETHERAND

IBMTRNET

IBMPCNET

SDLC

TWINAX (Not supported by Host Integration Server 2009)

X25DLC

adapter_num

The DLC adapter number for host link.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

sscp_lu_sess_state

Specifies the state of the SSCP-LU session.

lu_lu_sess_state

Specifies the state of the LU-LU session. The state can be one of the following:

AP_NOT_BOUND

The LU-LU session is not bound.

AP_BOUND

The LU-LU session is bound.

AP_BINDING

The LU-LU session is in the process of binding.

AP_UNBINDING

The LU-LU session is in the process of unbinding.

link_id

Name of local logical link station being used.

Returned by Host Integration Server
Members
lu_0_3_entry_len

Size of this LU entry.

access_type

The access type (AP_3270 or AP_LUA).

lu_type

The LU type (AP_LU0, AP_LU1, AP_LU2, or AP_LU3).

lu_daf

The network addressable unit of the LU for which the information is displayed.

lu_short_name

The 1 byte ASCII LU short name.

lu_long_name

The 8 byte ASCII LU long name.

session_id

The LU-LU session ID.

dlc_name

DLC name set to one of the following:

IBMTRNET

SDLC

TWINAX (Not supported by Host Integration Server 2009)

X25DLC

adapter_num

The DLC adapter number for host link. Always set to zero.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

sscp_lu_sess_state

Specifies the state of the SSCP-LU session.

lu_lu_sess_state

Specifies the state of the LU-LU session. The state can be one of the following:

AP_NOT_BOUND

The LU-LU session is not bound.

AP_BOUND

The LU-LU session is bound.

AP_BINDING

The LU-LU session is in the process of binding.

AP_UNBINDING

The LU-LU session is in the process of unbinding.

link_id

Name of connection.

System Default Information

System default information is defined or returned as described here.

Defined by IBM ES for OS/2 version 1.0
Members
default_mode_name

Mode name used for undefined mode name is sent or received.

default_local_lu_name

Alias or local default LU.

implicit_partner_lu_support

Indicates if implicit partner LU support is enabled.

maximum_held_alerts

Number of alerts that will be held by NS/2 if there is no active link to a focal point.

default_tp_conversation_security_rqd

Specifies if conversation security is used for default TPs.

maximum_mc_ll_send_size

Maximum length of a logical record used on a mapped conversation for sending data to either the inbound or outbound implicit remote LU.

directory_for_inbound_attaches

Name of OS/2 directory used by Attach Manager.

default_tp_operation

Set to one of the following:

QUEUED_OPERATOR_STARTED

QUEUED_OPERATOR_PRELOADED

QUEUED_AM_STARTED

NONQUEUED_AM_STARTED

default_tp_program_type

Set to one of the following:

BACKGROUND

FULL_SCREEN

PRESENTATION_MANAGER

VIO_WINDOWABLE

Returned by Host Integration Server
Members
default_mode_name

Always set to NULL.

default_local_lu_name

Always set to spaces.

implicit_partner_lu_support

Always set to NO.

maximum_held_alerts

Always set to zero.

default_tp_conversation_security_rqd

Always set to NO.

maximum_mc_ll_send_size

Always set to 16384.

directory_for_inbound_attaches

Always returned * and indicates that the current path should be used.

default_tp_operation

Always set to QUEUED_AM_STARTED.

default_tp_program_type

Always set to FULL_SCREEN.

LU 6.2 Definition Information

There are no differences for this information type.

Partner Definition Information

Partner definition information is defined or returned as described here.

Defined by IBM ES for OS/2 version 1.0

Members

maximum_mc_ll_send_size

Maximum length of a logical record used on a mapped conversation for sending data to the partner LU.

number_of_alternate_aliases

Specifies number of alternate aliases configured.

Returned by Host Integration Server

maximum_mc_ll_send_size

Always set to 16384.

number_of_alternate_aliases

Always set to zero.

Mode Definition Information

Mode definition information is defined or returned as described here.

Defined by IBM ES for OS/2 version 1.0

Members
cos_name

Name of class of service.

Returned by Host Integration Server

Members
cos_name

Set to EBCDIC spaces.

Link Definition Information

Link definition information is provided in the **link_def_info_sect** structure as defined below.

Syntax

```
typedef struct link_def_info_sect {
    unsigned long  link_def_init_sect_len;
    unsigned short num_link_def;
    unsigned short total_link_def;
} LINK_DEF_INFO_SECT;
```

Members

link_def_init_sect_len

The length of the initial link definition information section, including this parameter, up to the first link definition overlay group. The length does not include any previous information sections.

num_link_def

The number of link definitions returned by the **DISPLAY** verb into your program's buffer. This is the number of times the link definition overlay is repeated.

total_link_def

The total number of link definitions. This number is the same as the number returned in the **num_link_def** member except when APPC has more information about link definitions than it can place in the supplied buffer, in which case this number is larger.

For each link definition, a **link_def_overlay** structure for the link definition is provided as defined below.

Syntax

```
typedef struct link_def_overlay {
    unsigned long  link_def_entry_len;
    unsigned char  link_name[8];
    unsigned char  adj_fq_cp_name[17];
    unsigned char  adj_node_type;
    unsigned long  dlc_name[8];
    unsigned char  adapter_num;
    unsigned char  dest_addr_len;
    unsigned char  dest_addr[32];
    unsigned char  preferred_nn_server;
    unsigned char  auto_act_link;
    unsigned char  tg_number;
    unsigned char  lim_res;
    unsigned char  solicit_sscp_session;
    unsigned char  initself;
    unsigned char  bind_support;
    unsigned char  ls_role;
    unsigned char  line_type;
    unsigned long  eff_capacity;
    unsigned char  conn_cost;
    unsigned char  byte_cost;
    unsigned char  propagation_delay;
    unsigned char  user_def_1;
    unsigned char  user_def_2;
    unsigned char  user_def_3;
    unsigned char  security;
    unsigned char  reserv;
 } LINK_OVERLAY;
```

Defined by IBM ES for OS/2 version 1.0

Members

link_def_entry_len

Size of this link definition entry.

link_name

Local logical link station name (EBCDIC).

dlc_name

Data link control (DLC) name set to one of the following:

ETHERAND

IBMTRNET

IBMPCNET

SDLC

TWINAX (Not supported by Host Integration Server 2009)

X25DLC

adj_fq_cp_name

Fully qualified **cp_name** in adjacent node.

adj_node_type

The adjacent node type (AP_ADJACENT_NN, AP_LEARN, or AP_LEN).

adapter_num

DLC adapter number used by this link.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

cp_cp_sess_spt

Specifies whether the link supports CP-CP sessions.

preferred_nn_server

Indicates if this is the preferred NN server.

auto_act_link

Indicates if the link should be automatically activated.

tg_number

Transmission group number.

lim_res

Indicates if this is a limited resource.

solicit_sscp_session

Indicates whether to solicit an SSCP session.

initself

Indicates if the node supports receiving INIT_SELF over this link.

bind_support

Indicates whether BIND support is available.

ls_role

Specifies the link station role.

line_type

The line type.

eff_capacity

Highest bit rate transmission effective capacity supported.

conn_cost

Relative cost per connection time using this link.

byte_cost

Relative cost of transmitting a byte over link.

propagation_delay

Indicates amount of time for signal to travel length of link. Set to one of the following:

AP_PROP_DELAY_MINIMUM

AP_PROP_DELAY_LAN

AP_PROP_DELAY_TELEPHONE

AP_PROP_DELAY_PKT_SWITCHED_NET

AP_PROP_DELAY_SATELLITE

AP_PROP_DELAY_MAXIMUM

user_def_1

User-defined TG characteristics.

user_def_2

User-defined TG characteristics.

user_def_3

User-defined TG characteristics.

security

The security value for this link. Set to one of the following:

AP_SEC_NONSECURE

AP_SEC_PUBLIC_SWITCHED_NETWORK

AP_SEC_UNDERGROUND_CABLE

AP_SEC_SECURE_CONDUIT

AP_SEC_GUARDED_CONDUIT

AP_SEC_ENCRYPTED

AP_SEC_GUARDED_RADIATION

Returned by Host Integration Server
Members
link_def_entry_len

Size of this link definition entry.

link_name

Local logical link station name (EBCDIC).

dlc_name

Data link control (DLC) name set to one of the following:

IBMTRNET

SDLC

X25DLC

adj_fq_cp_name

Fully qualified **cp_name** in adjacent node. Always set to EBCDIC spaces.

adj_node_type

The adjacent node type. Always set to AP_LEN.

adapter_num

DLC adapter number used by this link. Always set to zero.

dest_addr_len

Length of the destination adapter address.

dest_addr

The destination adapter address.

cp_cp_sess_spt

Specifies whether the link supports CP-CP sessions. Always set to AP_NO.

preferred_nn_server

Indicates if this is the preferred NN server.

auto_act_link

Indicates if the link should be automatically activated.

tg_number

Transmission group number. Always set to zero.

lim_res

Indicates if this is a limited resource.

solicit_sscp_session

Indicates whether to solicit an SSCP session.

initself

Indicates if the node supports receiving INIT_SELF over this link.

bind_support

Indicates whether BIND support is available.

ls_role

Specifies the link station role.

line_type

The line type.

effective_capacity

Highest bit rate transmission effective capacity supported. Always set to zero.

conn_cost

Relative cost per connection time using this link. Always set to zero.

byte_cost

Relative cost of transmitting a byte over link. Always set to zero.

propagation_delay

Indicates amount of time for signal to travel length of link. Set to one of the following: Always set to AP_PROP_DELAY_MAXIMUM.

user_def_1

User-defined TG characteristics. Always set to zero.

user_def_2

User-defined TG characteristics. Always set to zero.

user_def_3

User-defined TG characteristics. Always set to zero.

security

The security value for this link. Always set to AP_SEC_NONSECURE.

Management Services Information

Information on management services is provided in the **ms_info_sect** structure as defined below.

Syntax

```
typedef struct ms_info_sect {
    unsigned long  ms_init_sect_len;
    unsigned char  held_mds_mu_alerts;
    unsigned char  held_nmvt_alerts;
    unsigned short num_fps;
    unsigned short total_fps;
    unsigned short num_ms_appls;
    unsigned short total_ms_appls;
    unsigned short num_act_trans;
    unsigned short total_act_trans;
} MS_INFO_SECT;
```

Members

ms_init_sect_len

The length of the initial MS information section, including this parameter, up to the first MS focal point group. The length does not include any previous information sections.

held_mds_mu_alerts

The number of management service MDS alerts being held that will be sent to the management service alert focal point (FP) when one becomes available.

held_nmvt_alerts

The number of management service NMVT alerts being held that will be sent to the management service alert focal point (FP) when one becomes available.

num_fps

The number of management service focal points (MS FPs) for which the information listed under MS Focal Point Group is returned. This is the number of times the information group is repeated.

total_fps

The total number of management service focal points for which APPC has information. This number is the same as the number returned in the **num_fps** member except when APPC has more information about management service focal points than it can place in the supplied buffer, in which case this number is larger.

num_ms_appls

The number of registered MS applications for which the information listed under Registered MS Application Group is returned. This is the number of times the information group is repeated.

total_ms_appls

The total number of registered MS applications for which APPC has information. This number is the same as the number returned in the **num_ms_appls** member except when APPC has more information about registered MS applications than it can place in the supplied buffer, in which case this number is larger.

num_act_trans

The number of MS active transactions for which the information listed under MS Active Transaction Group is returned. This is the number of times the information group is repeated.

total_act_trans

The number or MS active transactions for which APPC has information. This number is the same as the number returned in the **num_act_trans** member except when APPC has more information about registered MS active transactions than it can place in the supplied buffer, in which case this number is larger.

For each local and remote management service focal point group, an **ms_fp_overlay** structure for the focal point group is provided as defined below.

Syntax

```
typedef struct ms_fp_overlay {
    unsigned long  ms_fp_entry_len;
    unsigned char  ms_appl_name[8];
    unsigned char  ms_category[4];
    unsigned char  fp_fq_cp_name[17];
    unsigned char  bkup_appl_name[8];
    unsigned char  bkup_fp_fq_cp_name[17];
    unsigned char  reserv1;
    unsigned char  fp_type;
    unsigned char  fp_status;
    unsigned char  fp_routing;
} MS_FP_OVERLAY;
```

Members

ms_fp_entry_len

Size of this management service focal point information entry.

ms_appl_name

The management service application name of the current active focal point (EBCDIC).

ms_category

The management service category.

fp_fq_cp_name

The fully qualified control point name of the node on which the current (active) management service focal point is located (EBCDIC). If the local node has no focal point, a value of all EBCDIC space characters (0x40) is returned.

bkup_appl_name

The management service application name of the backup focal point, if one is known (EBCDIC).

bkup_fp_fq_cp_name

The fully qualified control point name of the node on which the backup management service focal point is located, if one is known (EBCDIC). If the local node has no backup focal point, a value of all EBCDIC space characters (0x40) is returned.

fp_type

The type of the focal point for the local management service entry point node. The focal point type depends on how the focal point-end point relationship was established, and on whether the local node is configured as an NN, EN, or LEN node (an EN without CP-CP session support). The type can be one of the following:

AP_EXPLICIT_PRIMARY_FP

The current focal point type is explicit primary.

AP_BACKUP_FP

The current focal point type is back up.

AP_DEFAULT_PRIMARY_FP

The current focal point type is default primary.

AP_DOMAIN_FP

The current focal point type is domain.

AP_HOST_FP

The current focal point type is host.

AP_NO_FP

Currently the local node has no focal point.

fp_status

The status of the management service focal point. The status can be one of the following:

AP_NOT_ACTIVE

The focal point has been acquired, but has since become unavailable.

AP_ACTIVE

The remote focal point has been acquired and is available.

AP_PENDING

A request has been sent to a remote primary or backup focal point to acquire that FP, and its reply has not yet been received.

AP_NEVER_ACTIVE

The focal point has never been acquired, but one or more registered management service applications have requested focal point information.

fp_routing

The routing used to send unsolicited requests to the management service focal point when the local node is an EN. Note that requests from an NN are always sent directly to the focal point.

The routing can be one of the following:

AP_DEFAULT

Unsolicited management service requests destined for the focal point are sent from the EN to its serving NN for forwarding to the focal point.

AP_DIRECT

Unsolicited management service requests destined for the focal point are sent directly to the focal point.

Remarks

When a program registers a management service application name, it can request focal point information. When APPC acquires the focal point, it passes the program the focal point information, which includes the type of routing to use to send unsolicited management service requests to the focal point.

APPC TP Verbs

This section describes the Advanced Program-to-Program Communications (APPC) transaction program (TP) verbs. The description of each verb provides:

- A definition of the verb.
- The structure defining the verb control block (VCB) used by the verb. The structure is contained in the WINAPPC.H file. The length of each VCB field is in bytes. Fields beginning with reserv (for example, reserv2) are reserved.
- The parameters (VCB fields) supplied to and returned by APPC. A description of each parameter is provided, along with its possible values and other information.
- The conversation state(s) in which the verb can be issued.
- The state(s) to which the conversation can change upon return from the verb. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the verb.

Most parameters supplied to and returned by APPC are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants, which are established by **#define** statements in the WINAPPC.H header file. For example, the **opcode** (operation code) member of the **mc_send_data** structure used by the **MC_SEND_DATA** verb is the hexadecimal value represented by the symbolic constant AP_M_SEND_DATA. Use only the symbolic constants when writing TPs.

In This Section

GET_TP_PROPERTIES

The **GET_TP_PROPERTIES** verb returns attributes of the transaction program (TP) and the current transaction.

The following structure describes the verb control block used by the **GET_TP_PROPERTIES** verb.

Syntax

```
struct get_tp_properties {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned char   tp_name[64];
    unsigned char   lu_alias[8];
    unsigned char   luw_id[26];
    unsigned char   fqlu_name[17];
    unsigned char   reserve3[10];
    unsigned char   user_id[10];
    unsigned char   prot_luw_id[26];
    unsigned char   pwd[10];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_GET_TP_PROPERTIES.

opext

Supplied parameter. Specifies the verb operation extension. If the AP_EXTD_VCB bit is set, this indicates that the **get_tp_properties** structure includes the **prot_luw_id** member used for Sync Point support. Otherwise the verb control block ends immediately after the **user_id** member.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

tp_name

Returned parameter. Supplies the TP name of the TP that issued the verb. The name is returned as a 64-byte EBCDIC string, padded on the right with EBCDIC spaces.

lu_alias

Returned parameter. Supplies the alias name assigned to the local LU. It is returned as an 8-byte ASCII string padded on the right with ASCII spaces.

luw_id

Returned parameter. Supplies the unprotected logical unit-of-work identifier for the transaction in which the TP is participating. Several TPs can be involved in a transaction. This identifier, which is assigned on behalf of the TP initiating the

transaction, allows the conversation that makes up the transaction to be logically connected.

The **luw_id** can be represented as an **luw_id_overlay** structure with the following fields:

```
typedef struct luw_id_overlay { unsigned char fqla_name_len; unsigned char fqla_name[17]; nsigned char
instance[6]; unsigned char sequence[2];} LUW_ID_OVERLAY;
```

luw_id.fqla_name_len

A 1-byte length of the fully qualified LU name for the LU of the originating TP.

luw_id.fqla_name

The fully qualified name of the LU for the originating TP. The name is returned as a 17-byte EBCDIC string, consisting of the NETID, a period, and the LU name. If the length of the name is fewer than 17 bytes, the **instance** and **sequence** numbers follow immediately. (Note that because of this, you should not use the fields of the **luw_id_overlay** structure to access those values. These are provided for compatibility only.)

luw_id.instance

A 6-byte string uniquely generated by the LU for the originating TP.

luw_id.sequence

A 2-byte number that indicates the segment of unit-of-work. (This is always set to 1, if Sync Point is not supported.)

If the **luw_id** length is fewer than 26 bytes, it is padded on the right with EBCDIC spaces.

fqlu_name

Returned parameter. Supplies the fully qualified name of the local LU. The name is returned as a 17-byte EBCDIC string, consisting of the NETID, a period, and the LU name. The name is padded on the right with EBCDIC spaces.

reserve3

A reserved field.

user_id

Supplied parameter. Indicates the **user_id** supplied by the initiating TP in the allocation request. The name is supplied as a 10-byte EBCDIC string, padded on the right with EBCDIC spaces.

prot_luw_id

Returned parameter. Contains the protected logical unit-of-work identifier for the transaction in which the TP is participating, if the conversation was allocated with **synclevel** Sync Point.

Several TPs can be involved in a transaction. This identifier, which is assigned on behalf of the TP initiating the transaction, allows the conversation that makes up the transaction to be logically connected.

The **prot_luw_id** can be represented as an **luw_id_overlay** structure with the following fields:

```
typedef struct luw_id_overlay { unsigned char fqla_name_len; unsigned char fqla_name[17]; nsigned char
instance[6]; unsigned char sequence[2];} LUW_ID_OVERLAY;
```

luw_id.fqla_name_len

A 1-byte length of the fully qualified LU name for the LU of the originating TP.

luw_id.fqla_name

The fully qualified name of the LU for the originating TP. The name is returned as a 17-byte EBCDIC string, consisting of the NETID, a period, and the LU name. If the length of the name is fewer than 17 bytes, the **instance** and **sequence** numbers follow immediately. (Note that because of this, you should not use the fields of the **luw_id_overlay** structure to access those values. These are provided for compatibility only).

luw_id.instance

A 6-byte string uniquely generated by the LU for the originating TP.

luw_id.sequence

A 2-byte number that indicates the segment of unit-of-work. (This is always set to 1, if Sync Point is not supported.)

If the **prot_luw_id** length is fewer than 26 bytes, it is padded on the right with EBCDIC spaces.

pwd

Supplied parameter. Contains the password of the **user_id** of the initiating TP in the allocation request. The password is supplied as a 10-byte EBCDIC string, padded on the right with EBCDIC spaces.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_TP_BUSY

Primary return code; the local TP has issued a call to APPC while APPC was processing another call for the same TP. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **tp_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

This verb relates to the TP rather than a specification conversation, so the TP can issue the verb in any state. There is no state change.

The **luw_id** member contains fields for **fqla_name_len** (the length of the fully qualified LU name of the LU originating the TP), **fqla_name** (the fully qualified name of the LU originating the TP), **instance** (generated uniquely by the LU originating the TP), and **sequence** (always set to 1 and indicating the segment of unit-of-work).

SET_TP_PROPERTIES

The **SET_TP_PROPERTIES** verb enables a transaction program (TP) to set its logical unit-of-work identifiers (LUWIDs) to either an existing value, by providing the LUWIDs, or request that the SNA server generate new ones and use them from then on. When the LUWID is generated by the SNA server, it is guaranteed to be unique. This verb is used only if Sync Point support is enabled.

The following structure describes the verb control block (VCB) used by the **SET_TP_PROPERTIES** verb.

Syntax

```
struct set_tp_properties {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned char   set_prot_id;
    unsigned char   new_prot_id;
    unsigned char   prot_id[26];
    unsigned char   set_unprot_id;
    unsigned char   new_unprot_id;
    unsigned char   unprot_id[26];
    unsigned char   set_user_id;
    unsigned char   reserv3;
    unsigned char   user_id[10];
    unsigned char   reserv4[10];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_SET_TP_PROPERTIES.

opext

Supplied parameter. Specifies the verb operation extension. The AP_EXTD_VCB bit must be set to indicate that the **set_tp_properties** structure requires Sync Point support.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

set_prot_id

Supplied parameter. Indicates whether the **prot_id** member should be modified. Legal values are AP_YES or AP_NO.

new_prot_id

Supplied parameter. Indicates whether Microsoft® Host Integration Server should use the supplied **prot_id** LUWID member or create a new LUWID. Legal values are AP_YES (create a new LUWID) or AP_NO (use the supplied LUWID).

prot_id

This member is the protected logical unit-of-work identifier for the transaction in which the TP is participating. It is ignored if **set_prot_id** is AP_NO. It is a supplied parameter if **new_unprot_id** is AP_NO or a returned parameter if **new_unprot_id** is AP_YES.

Several TPs can be involved in a transaction. This identifier, which is assigned on behalf of the TP initiating the transaction, allows the conversation that makes up the transaction to be logically connected.

The **prot_id** can be represented as an **luw_id_overlay** structure with the following fields:

```
typedef struct luw_id_overlay { unsigned char fqla_name_len; unsigned char fqla_name[17]; nsigned char
instance[6]; unsigned char sequence[2];} LUW_ID_OVERLAY;
```

luw_id.fqla_name_len

A 1-byte length of the fully qualified LU name for the LU of the originating TP.

luw_id.fqla_name

The fully qualified name of the LU for the originating TP. The name is returned as a 17-byte EBCDIC string, consisting of the NETID, a period, and the LU name. If the length of the name is fewer than 17 bytes, the **instance** and **sequence** numbers follow immediately. (Note that because of this, you should not use the fields of the **luw_id_overlay** structure to access those values. These are provided for compatibility only).

luw_id.instance

A 6-byte string uniquely generated by the LU for the originating TP.

luw_id.sequence

A 2-byte number that indicates the segment of unit-of-work. (This is always set to 1 if Sync Point is not supported.)

If the **luw_id** length is fewer than 26 bytes, it is padded on the right with EBCDIC spaces.

set_unprot_id

Supplied parameter. Indicates whether the **unprot_id** member should be modified. Legal values are AP_YES or AP_NO.

new_unprot_id

Supplied parameter. Indicates whether Host Integration Server should use the supplied **unprot_id** LUWID member or create a new LUWID. Legal values are AP_YES (create a new LUWID) or AP_NO (use the supplied LUWID).

unprot_id

This member is the unprotected logical unit-of-work identifier for the transaction in which the TP is participating. It is ignored if **set_unprot_id** is AP_NO. It is a supplied parameter if **new_unprot_id** is AP_NO or a returned parameter if **new_unprot_id** is AP_YES.

Several TPs can be involved in a transaction. This identifier, which is assigned on behalf of the TP initiating the transaction, allows the conversation that makes up the transaction to be logically connected.

The **prot_id** can be represented as an **luw_id_overlay** structure with the following fields:

```
typedef struct luw_id_overlay { unsigned char fqla_name_len; unsigned char fqla_name[17]; nsigned char
instance[6]; unsigned char sequence[2];} LUW_ID_OVERLAY;
```

luw_id.fqla_name_len

A 1-byte length of the fully qualified LU name for the LU of the originating TP.

luw_id.fqla_name

The fully qualified name of the LU for the originating TP. The name is returned as a 17-byte EBCDIC string, consisting of the NETID, a period, and the LU name. If the length of the name is fewer than 17 bytes, the **instance** and **sequence** numbers follow immediately. (Note that because of this, you should not use the fields of the **luw_id_overlay** structure to access those values. These are provided for compatibility only.)

luw_id.instance

A 6-byte string uniquely generated by the LU for the originating TP.

luw_id.sequence

A 2-byte number that indicates the segment of unit-of-work. (This is always set to 1 if Sync Point is not supported.)

If the **luw_id** length is fewer than 26 bytes, it is padded on the right with EBCDIC spaces.

set_user_id

Supplied parameter. Indicates whether the **user_id** member should be modified. Legal values are AP_YES or AP_NO.

reserve3

A reserved field.

user_id

Supplied parameter. Indicates the **user_id** that should be used by the initiating TP in the allocation request. The name is a 10-byte EBCDIC string, padded on the right with EBCDIC spaces. This parameter is ignored if **set_user_id** is AP_NO.

reserve4

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_TP_BUSY

Primary return code; the local TP has issued a call to APPC while APPC was processing another call for the same TP. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **tp_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

This verb relates to the TP rather than a specification conversation, so the TP can issue the verb in any state. There is no state change.

The **prot_id** and **unprot_id** members contain fields for **fqla_name_len** (the length of the fully qualified LU name of the LU

originating the TP), **fqla_name** (the fully qualified name of the LU originating the TP), **instance** (generated uniquely by the LU originating the TP), and **sequence** (always set to 1 and indicating the segment of unit-of-work).

It is the responsibility of the application (the Sync Point support component) to transmit the new LUWID PS header to the partner Sync Point support when the protected LUWID is changed. Similarly, when the new LUWID PS header is received, the application must inform the LU by issuing a **SET_TP_PROPERTIES** verb.

TP_ENDED

The **TP_ENDED** verb is issued by both the invoking and invoked transaction program (TP), and notifies APPC that the TP is ending.

For the Microsoft® Windows® version 3.*x* system, it is recommended that you use the WinAsyncAPPC function rather than the blocking version of this call.

The following structure describes the verb control block (VCB) used by the **TP_ENDED** verb.

Syntax

```
struct tp_ended {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned char   type;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_TP_ENDED.

opext

Supplied parameter. Specifies the verb operation extension. This field is not used by the **TP_ENDED** verb.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

type

Supplied parameter. Specifies the type of termination to be performed. The following are allowed values:

- AP_HARD indicates that all active verbs for the TP are terminated; the session(s) being used by the conversation(s) are ended. Both the local TP and the partner TP can receive conversation failure return codes (AP_DEALLOC_ABEND for mapped conversations and AP_DEALLOC_ABEND_PROG for basic conversations).

- AP_SOFT indicates that the TP waits for all active verbs to complete; the session being used by the conversation remains active.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_TP_ID

Secondary return code; APPC did not recognize the **tp_id** as an assigned TP identifier.

AP_BAD_TYPE

Secondary return code; the specified **type** value was not recognized by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_TP_BUSY

Primary return code; the local TP has issued a call to APPC while APPC was processing another call for the same TP. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **tp_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

In response to **TP_ENDED**, APPC frees the resources used by the TP. After this verb executes, the TP identifier is no longer valid; the TP cannot issue any more APPC conversation verbs.

The conversation can be in any state when the TP issues this verb.

If the conversation is in SEND state, **TP_ENDED** performs the function of DEALLOCATE or MC_DEALLOCATE with **dealloc_type** set to AP_FLUSH.

If the conversation is in a state other than RESET or SEND, **TP_ENDED** performs the function of **DEALLOCATE** or **MC_DEALLOCATE** with **dealloc_type** set to AP_ABEND (for a mapped conversation) or AP_ABEND_PROG (for a basic conversation).

After successful execution (**primary_rc** is AP_OK), there is no APPC state.

TP_STARTED

The **TP_STARTED** verb is issued by the invoking transaction program (TP), and notifies APPC that the TP is starting.

For the Microsoft® Windows® version 3.*x* system, it is recommended that you use the WinAsyncAPPC function rather than the blocking version of this call.

The following structure describes the verb control block used by the **TP_STARTED** verb.

Syntax

```
struct tp_started {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   lu_alias[8];
    unsigned char   tp_id[8];
    unsigned char   tp_name[64];
    unsigned char   syncpoint_rqd;
};
```

Remarks

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_TP_STARTED.

opext

Supplied parameter. Specifies the verb operation extension. If the AP_EXTD_VCB bit is set, this indicates that the **tp_started** structure includes the **syncpoint_rqd** member used for Sync Point support. Otherwise, the verb control block ends immediately after the **tp_name** member.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

lu_alias

Supplied parameter. Specifies the alias by which the local LU is known to the local TP.

The name must match an LU alias established during configuration. APPC checks the LU alias against the current Host Integration Server configuration file. Due to the client/server architecture used by Host Integration Server, however, this parameter is not validated until an ALLOCATE or MC_ALLOCATE is performed.

This parameter is an 8-byte ASCII character string. It can consist of the following ASCII characters:

- Uppercase letters
- Numerals from 0 through 9
- Spaces
- Special characters $, #, % and @

The first character of this string cannot be a space.

If the value of this parameter is fewer than eight bytes in length, pad it on the right with ASCII spaces (0x20).

To use an LU from the default LU pool, set this field to eight hexadecimal zeros. For more information, see Default LUs.

tp_id

Returned parameter. Identifies the newly established TP.

tp_name

Supplied parameter. Specifies the name of the local TP.

Under the Host Integration Server implementation of APPC, this parameter is ignored when issued by **TP_STARTED**. However, this parameter is required if the program runs under the IBM ES for OS/2 version 1.0 implementation of APPC.

This parameter is a 64-byte EBCDIC character string and is case-sensitive. The **tp_name** parameter can consist of the following EDCDIC characters:

- Uppercase and lowercase letters
- Numerals from 0 through 9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes in length, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention for a service TP name is up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F.

syncpoint_rqd

This optional parameter is only applicable if the AP_EXTD_VCB bit is set in the **opext** parameter and Sync Point services are required.

- AP_YES if Sync Point is required.
- AP_NO if Sync Point is not required.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_TP_BUSY

Primary return code; the local TP has issued a call to APPC while APPC was processing another call for the same TP.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

In response to **TP_STARTED**, APPC generates a TP identifier for the invoking TP. This identifier is a required parameter for subsequent APPC verbs issued by the invoking TP.

This must be the first APPC verb issued by the invoking TP. Consequently, no prior APPC state exists.

If the verb executes successfully (**primary_rc** is AP_OK), the state changes to RESET.

In This Section

- Default LUs

Default LUs

Any LU can be configured to be in a pool of default local LUs available for use by invoking transaction programs (TPs).

For a user or group who will be using TPs, 5250 emulators, and/or APPC applications, you can assign a default local APPC LU and a default remote APPC LU. If the invoking TP specifies the LU alias that it uses (in TP_STARTED), that LU alias must match a local APPC LU alias on the supporting SNA server. If the invoking TP leaves the LU alias blank in **TP_STARTED**, one of two methods for designating a default LU must be carried out on the supporting SNA server:

- Assign a default local APPC LU to the user or group that starts the invoking TP (that is, the user or group logged on at the system from which **TP_STARTED** is issued).

  —or—

- Designate one or more LUs as members of the default outgoing local APPC LU pool. The SNA server first attempts to determine the default local APPC LU of the associated user or group, then attempts to assign an available LU from the default outgoing local APPC LU pool; if these attempts fail, the SNA server rejects the request.

AS/400 Environment

In AS/400 environments, the ability to assign default APPC LUs to users or groups is especially useful because it gives the administrator centralized control over these LU assignments. In such environments, for each user or group, assign both a default local APPC LU and a default remote APPC LU. Assigning a default local APPC LU for each user fulfills the normal AS/400 procedure of assigning local LUs on a per-user basis. Assigning a default remote APPC LU is equivalent to assigning a default AS/400 for the user to connect to, since the remote LU designates the AS/400. By making these assignments, the administrator can centrally control the default AS/400 that a 5250 emulator user connects to.

See Also

Reference

TP_STARTED

APPC Conversation Verbs

This section describes the Advanced Program-to-Program Communications (APPC) conversation verbs. The description of each verb provides:

- A definition of the verb.
- The structure defining the verb control block (VCB) used by the verb. The structure is contained in the WINAPPC.H file. The length of each VCB field is in bytes. Fields beginning with reserv (for example, reserv2) are reserved.
- The parameters (VCB fields) supplied to and returned by APPC. A description of each parameter is provided, along with its possible values and other information.
- The conversation state(s) in which the verb can be issued.
- The state(s) to which the conversation can change upon return from the verb. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the verb.

Mapped conversation verbs are preceded by an **MC_** designator. For example, the mapped conversation verb **MC_ALLOCATE** corresponds to the basic conversation verb **ALLOCATE**.

Most parameters supplied to and returned by APPC are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants, which are established by **#define** statements in the WINAPPC.H header file. For example, the **opcode** (operation code) member of the **mc_send_data** structure used by the **MC_SEND_DATA** verb is the hexadecimal value represented by the symbolic constant AP_M_SEND_DATA. Use only the symbolic constants when writing transaction programs (TPs).

In This Section

- MC_FLUSH
- MC_GET_ATTRIBUTES
- MC_POST_ON_RECEIPT
- MC_PREPARE_TO_RECEIVE
- MC_RECEIVE_AND_POST
- MC_RECEIVE_AND_WAIT
- MC_RECEIVE_IMMEDIATE
- MC_RECEIVE_LOG_DATA
- MC_REQUEST_TO_SEND
- MC_SEND_CONVERSATION
- MC_SEND_DATA
- MC_SEND_ERROR
- MC_TEST_RTS
- MC_TEST_RTS_AND_POST
- POST_ON_RECEIPT
- PREPARE_TO_RECEIVE
- RECEIVE_ALLOCATE
- RECEIVE_ALLOCATE_EX
- RECEIVE_ALLOCATE_EX_END
- RECEIVE_AND_POST
- RECEIVE_AND_WAIT
- RECEIVE_IMMEDIATE
- RECEIVE_LOG_DATA
- REQUEST_TO_SEND
- SEND_CONVERSATION
- SEND_DATA
- SEND_ERROR
- TEST_RTS

- TEST_RTS_AND_POST

ALLOCATE

The **ALLOCATE** verb is issued by the invoking transaction program (TP). It allocates a session between the local logical unit (LU) and partner LU and (in conjunction with RECEIVE_ALLOCATE) establishes a conversation between the invoking TP and the invoked TP. After this verb executes successfully, APPC generates a conversation identifier (**conv_id**). The **conv_id** is a required parameter for all other APPC conversation verbs.

The following structure describes the verb control block used by the **ALLOCATE** verb.

Syntax

```
struct allocate {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    conv_type;
    unsigned char    synclevel;
    unsigned char    reserv3[2];
    unsigned char    rtn_ctl;
    unsigned char    reserv4;
    unsigned long    conv_group_id;
    unsigned long    sense_data;
    unsigned char    plu_alias[8];
    unsigned char    mode_name[8];
    unsigned char    tp_name[64];
    unsigned char    security;
    unsigned char    reserv5[11];
    unsigned char    pwd[10];
    unsigned char    user_id[10];
    unsigned short   pip_dlen;
    unsigned char FAR * pip_dptr;
    unsigned char    reserv7;
    unsigned char    fqplu_name[17];
    unsigned char    reserv8[8];
    unsigned long    proxy_user;
    unsigned long    proxy_domain;
    unsigned char    reserv9[16];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_ALLOCATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION. If the AP_EXTD_VCB bit is set, this indicates that an extended version of the verb control block is used. In this case, the **ALLOCATE** structure includes Sync Point support or privileged proxy feature support.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED.

conv_id

Returned parameter. Identifies the conversation established between the two TPs.

conv_type

Supplied parameter. Used only by **ALLOCATE** to specify the type of conversation to allocate and is either AP_BASIC_CONVERSATION or AP_MAPPED_CONVERSATION.

If **ALLOCATE** establishes a mapped conversation, the local TP must issue basic-conversation verbs and provide its own mapping layer to convert data records to logical records and logical records to data records. The partner TP can issue basic-conversation verbs and provide the mapping layer, or it can use mapped-conversation verbs (if the partner TP is using an implementation of APPC that supports mapped-conversation verbs). For more information, see your IBM SNA manual(s).

synclevel

Supplied parameter. Specifies the synchronization level of the conversation. It determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

- AP_NONE specifies that confirmation processing will not be used in this conversation.
- AP_CONFIRM_SYNC_LEVEL specifies that the TPs can use confirmation processing in this conversation.
- AP_SYNCPT specifies that TPs can use Sync Point Level 2 confirmation processing in this conversation.

reserv3

A reserved field.

rtn_ctl

Supplied parameter. Specifies when the local LU, acting on a session request from the local TP, should return control to the local TP. For information about sessions, see About Transaction Programs.

- AP_IMMEDIATE specifies that the LU allocates a contention-winner session, if one is immediately available, and returns control to the TP.
- AP_WHEN_SESSION_ALLOCATED specifies that the LU does not return control to the TP until it allocates a session or encounters one of the errors documented in Return Codes in this topic. (If the session limit is zero, the LU returns control immediately.) If a session is not available, the TP waits for one.
- AP_WHEN_SESSION_FREE specifies that the LU allocates a contention-winner or contention-loser session, if one is available or able to be activated, and returns control to the TP. If an error occurs, (as documented in Return Codes in this topic) the call will return immediately with the error in the **primary_rc** and **secondary_rc** fields.
- AP_WHEN_CONWINNER_ALLOCATED specifies that the LU does not return control until it allocates a contention-winner session or encounters one of the errors documented in Return Codes in this topic. (If the session limit is zero, the LU returns control immediately.) If a session is not available, the TP waits for one.
- AP_WHEN_CONV_GROUP_ALLOCATED specifies that the LU does not return control to the TP until it allocates the session specified by **conv_group_id**or encounters one of the errors documented in Return Codes in this topic. If a session is not available, the TP waits for it to become free.

Note
AP_IMMEDIATE is the only value for **rtn_ctl** that never causes a new session to start. For values other than AP_IMMEDIATE, if an appropriate session is not immediately available, Host Integration Server 2009 tries to start one. This causes the on-demand connection to be activated.

reserv4

A reserved field.

conv_group_id

Supplied/returned parameter. Specifies the identifier of the conversation group from which the session should be allocated. The **conv_group_id** is required only if **rtn_ctl**is set to WHEN_CONV_GROUP_ALLOC. When**rtn_ctl** specifies a different value and the **primary_rc** is AP_OK, this is a returned value.

sense_data

Returned parameter. Indicates an allocation error (retry or no-retry) and contains sense data.

plu_alias

Supplied parameter. Specifies the alias by which the partner LU is known to the local TP.

The **plu_alias** must match the name of a partner LU established during configuration.

The parameter is an 8-byte ASCII character string. It can consist of the following ASCII characters:

- Uppercase letters
- Numerals 0 through 9
- Spaces
- Special characters $, #, %, and @

The first character of this string cannot be a space.

If the value of this parameter is fewer than eight bytes, pad it on the right with ASCII spaces (0x20).

If you want to specify the partner LU with the **fqplu_name** parameter, fill this parameter with binary zeros.

For a user or group using TPs, 5250 emulators, and/or APPC applications, the system administrator can assign default local and remote LUs. In this case, the field is left blank or null and the default LUs are accessed when the user or group member starts an APPC program. For more information on default LUs, see Microsoft Host Integration Server 2009 Help.

mode_name

Supplied parameter. Specifies the name of a set of networking characteristics defined during configuration.

The value of **mode_name** must match the name of a mode associated with the partner LU during configuration.

The parameter is an 8-byte EBCDIC character string. It can consist of characters from the type A EBCDIC character set:

- Uppercase letters
- Numerals 0 through 9
- Special characters $, #, and @

The first character in the string must be an uppercase letter or a special character.

Do not use SNASVCMG in a mapped conversation. SNASVCMG is a reserved **mode_name** used internally by APPC. Using this name in a basic conversation is not recommended.

tp_name

Supplied parameter. Specifies the name of the invoked TP. The value of **tp_name** specified by **ALLOCATE** in the invoking TP must match the value of **tp_name** specified by RECEIVE_ALLOCATE in the invoked TP.

The parameter is a 64-byte EBCDIC character string and is case-sensitive. The **tp_name** parameter can consist of the following EBCDIC characters:

- Uppercase and lowercase letters

- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If **tp_name** is fewer than 64 bytes, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention is that a service TP name can have up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the type AE EBCDIC character set.

security

Supplied parameter. Provides the information that the partner LU requires to validate access to the invoked TP.

Based on the conversation security established for the invoked TP during configuration, use one of the following values:

- AP_NONE for an invoked TP that uses no conversation security.
- AP_PGM for an invoked TP that uses conversation security and thus requires a user identifier and password. Supply this information through the **user_id** and **pwd** parameters.
- AP_PROXY_PGM for an invoked TP with privileged proxy that uses conversation security and thus requires a user identifier and password. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields.
- AP_PROXY_SAME for a TP that has been invoked using privileged proxy with a valid user identifier and password supplied by the proxy, which in turn invokes another TP. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields.

For example, assume that TP A invokes TP B with a valid user identifier and password supplied by the privileged proxy, and TP B in turn invokes TP C. If TP B specifies the value AP_PROXY_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator tells TP C to not require the password (if TP C is configured to accept an already-verified indicator).

- AP_PROXY_STRONG for an invoked TP with privileged proxy that uses conversation security and thus requires a user identifier and password provided by the privileged proxy mechanism. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields. AP_PROXY_STRONG differs from AP_PROXY_PGM in that AP_PROXY_STRONG does not allow clear-text passwords. If the remote system does not support encrypted passwords (strong conversation security), then this call fails.
- AP_SAME for a TP that has been invoked with a valid user identifier and password, which in turn invokes another TP.

For example, assume that TP A invokes TP B with a valid user identifier and password, and TP B in turn invokes TP C. If TP B specifies the value AP_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator tells TP C to not require the password (if TP C is configured to accept an already-verified indicator).

When AP_SAME is used in an **ALLOCATE** verb, your application must always provide values for the **user_id** and **pwd** parameters in the verb control block. Depending on the properties negotiated between the SNA server and the peer LU, the **ALLOCATE** verb will send one of three kinds of Attach (FMH-5) messages, in this order of precedence:

1. If the LUs have negotiated "already verified" security, then the Attach sent by the SNA server will not include the contents of the **pwd** parameter field specified in the VCB.
2. If the LUs have negotiated "persistent verification" security, then the Attach sent by the SNA server will include the **pwd** parameter specified in the VCB, but only when the Attach is the first for the specified **user_id** parameter since the start of the LU-LU session, and will omit the **pwd** parameter on all subsequent Attaches (issued by your application or any other application using this LU-LU-mode triplet).

3. If the LUs have not negotiated either of the above, then the Attach sent by the SNA server will omit both the **user_id** and **pwd** parameters on all Attaches.

   Your application cannot tell which mode of security has been negotiated between the LUs, nor can it tell whether the **ALLOCATE** verb it is issuing is the first for that LU-LU-mode triplet. So your application must always set the **user_id** and **pwd** parameter fields in the VCB when **security** is set to AP_SAME.

   For more information on persistent verification and already verified security, see the SNA Formats Guide, section "FM Header 5: Attach (LU 6.2)".

- AP_STRONG for an invoked TP that uses conversation security and thus requires a user identifier and password. Supply this information through the **user_id** and **pwd** parameters. AP_STRONG differs from AP_PGM in that AP_STRONG does not allow clear-text passwords. If the remote system does not support encrypted passwords (strong conversation security), then this call fails.

  If the APPC automatic logon feature is to be used, **security** must be set to AP_PGM. See the Remarks section for details.

reserv5

A reserved field.

pwd

Supplied parameter. Specifies the password associated with **user_id**.

The **pwd** parameter is required only if **security** is set to AP_PGM or AP_SAME. It must match the password for **user_id** that was established during configuration.

The **pwd** parameter is a 10-byte EBCDIC character string and is case-sensitive. It can consist of the following EBCDIC characters:

- Uppercase and lowercase letters
- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

If the APPC automatic logon feature is to be used, the **pwd** character string must be hard-coded to MS$SAME. See the Remarks section for details.

user_id

Supplied parameter. Specifies the user identifier required to access the partner TP. It is required only if the security parameter is set to AP_PGM or AP_SAME.

The **user_id** parameter is a 10-byte EBCDIC character string and is case-sensitive. It must match one of the user identifiers configured for the partner TP.

The parameter can consist of the following EBCDIC characters:

- Uppercase and lowercase letters
- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If **user_id** is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

If the APPC automatic logon feature is to be used, the **user_id** character string must be hard-coded to MS$SAME. See the Remarks section for details.

pip_dlen

Supplied parameter. Specifies the length of the program initialization parameters (PIP) to be passed to the partner TP. The range is from 0 through 32767.

pip_dptr

Supplied parameter. Specifies the address of the buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

PIP data can consist of initialization parameters or environmental setup information required by a partner TP or remote operating system. The PIP data must follow the general data stream (GDS) format. For more information, see *SNA LU6.2 Reference: Peer Protocols* published by IBM.

For Microsoft Windows® 2000, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

reserv7

A reserved field.

fqplu_name

Supplied parameter. Specifies the fully qualified name of the partner LU. This must match the fully qualified name of the local LU defined in the remote node. The parameter consists of two type A EBCDIC character strings for the NETID and the LU name of the partner LU. The names are separated by an EBCDIC period (.).

This name must be provided if no **plu_alias** is specified. It can consist of the following EBCDIC characters:

- Uppercase letters
- Numerals 0 through 9
- Special characters $, #, and @

If the value of this parameter is fewer than 17 bytes, pad it on the right with EBCDIC spaces (0x40).

reserv8

A reserved field.

proxy_user

Supplied parameter. Specifies a LPWSTR pointing to a Unicode string containing the user name to be impersonated using the privileged proxy feature. This field can only be used when the AP_EXTD_VCB bit is set on the **opext** field, indicating an extended VCB.

proxy_domain

Supplied parameter. Specifies a LPWSTR pointing to a Unicode string containing the domain name of the user to be impersonated using the privileged proxy feature. This field can only be used when the AP_EXTD_VCB bit is set on the **opext** field, indicating an extended VCB.

reserv9

A reserved field.

Return Codes
AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; the supplied parameter **rtn_ctl** specified immediate (AP_IMMEDIATE) return of control to the TP, and the local LU did not have an available contention-winner session.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_RETURN_CONTROL

Secondary return code; the value specified for **rtn_ctl** was invalid.

AP_BAD_SECURITY

Secondary return code; the value specified for **security** was invalid.

AP_BAD_SYNC_LEVEL

Secondary return code; the value specified for **sync_level** was invalid.

AP_BAD_TP_ID

Secondary return code; the value specified for **tp_id** was invalid.

AP_PIP_LEN_INCORRECT

Secondary return code; the value of **pip_dlen** was greater than 32767.

AP_UNKNOWN_PARTNER_MODE

Secondary return code; the value specified for **mode_name** was invalid.

AP_BAD_PARTNER_LU_ALIAS

Secondary return code; APPC did not recognize the supplied **partner_lu_alias**.

AP_BAD_CONV_TYPE (for a basic conversation)

Secondary return code; the value specified for **conv_type** was invalid.

AP_NO_USE_OF_SNASVCMG (for a mapped conversation)

Secondary return code; SNASVCMG is not a valid value for **mode_name**.

AP_INVALID_DATA_SEGMENT

Secondary return code; the PIP data was longer than the allocated data segment, or the address of the PIP data buffer was wrong.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after **ALLOCATE**.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **ALLOCATE**, it can indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for system configured with multiple nodes using Host Integration Server, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

ALLOCATE can establish either a basic or mapped conversation.

The conversation state is RESET when the TP issues this verb. After successful execution (**primary_rc** is AP_OK), the state changes to SEND. If the verb does not execute, the state remains unchanged.

Several parameters of **ALLOCATE** are EBCDIC or ASCII strings. A TP can use the common service verb (CSV) CONVERT to translate a string from one character set to the other.

To send the **ALLOCATE** request immediately, the invoking TP can issue FLUSH or CONFIRM immediately after **ALLOCATE**. Otherwise, the **ALLOCATE** request accumulates with other data in the local LU's send buffer until the buffer is full.

By issuing **CONFIRM** after **ALLOCATE**, the invoking TP can immediately determine whether the allocation was successful (if **synclevel** is set to AP_CONFIRM_SYNC_LEVEL).

Normally, the value of the **ALLOCATE** verb's **mode_name** parameter must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU.

If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**.

Host Integration Server supports a feature called password substitution. This is a security feature supported by the latest version of the OS/400 operating system (V3R1) which encrypts any password that flows between two nodes on an Attach message. A password flows on an Attach whenever someone invokes an APPC transaction program specifying a user identifier and password. For example, this happens whenever anyone logs on to an AS/400.

Support for password substitution is indicated by setting bit 5 in byte 23 of the BIND request to 1 (which indicates that password substitution is supported). If the remote system sets this bit in the BIND response, the SNA server automatically encrypts the LU 6.2 conversation security password included in the FMH-5 Attach message. APPC applications using Host Integration Server 2009 automatically take advantage of this feature by setting the security field of the VCB to AP_PGM or AP_STRONG in the **ALLOCATE** request.

If an APPC application wants to force an encrypted password to flow, the application can specify AP_STRONG for the security field in the VCB in the **ALLOCATE** request. This option is implemented as defined in OS/400 V3R1, and is documented in the OS/400 CPI-C programmer reference as CM_SECURITY_PROGRAM_STRONG, where the LU 6.2 **pwd** (password) field is encrypted before it flows over the physical network.

The password substitution features is currently only supported by OS/400 V3R1 or later. If the remote system does not support this feature, the SNA server will UNBIND the session with the sense code of 10060006. The two nodes negotiate whether or not they support this feature in the BIND exchange. Host Integration Server sets a bit in the BIND, and also adds some random data on the BIND for encryption. If the remote node supports password substitution, it sets the same bit in the BIND response, and adds some (different) random data for decryption.

Host Integration Server 2009 supports automatic logon for APPC applications. This feature requires specific configuration by the network administrator: The APPC application must be invoked on the LAN side from a client of Host Integration Server 2009. The client must be logged into a Windows 2000 domain, but the client can be running on any operating system supported by the Host Integration Server 2009 APPC APIs.

The client application is coded to use "program" level security, with a special hard-coded APPC user name MS$SAME and password MS$SAME. When this session allocation flows from client to SNA server, the server looks up the host account and password corresponding to the Windows 2000 account under which the client is logged in, and substitutes the host account information into the APPC attach message it sends to the host.

Note
It is illegal for the remote node to set the bit specifying password substitution and not add the random data.

According to IBM, there are implementations of LU 6.2 password substitution that do not support password substitution but do echo the password substitution bit back to Host Integration Server 2009, without specifying any random data. When they do this, the SNA server will UNBIND the session with the sense code 10060006.This sense code is interpreted as:

- 1006 = Required field or parameter missing.
- 0006 = A required subfield of a control vector was omitted.

Host Integration Server should also log an Event 17 (APPC session activation failure: BIND negative response sent).

The correct solution is for the failing implementation to be fixed. However, as a short-term workaround, the following Host Integration Server service registry setting can be set:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\snaservr\parameters\NOPWDSUB: REG_SZ: YES

When this parameter is specified in the registry, password substitution support will be disabled.

Several updates have been made to Host Integration Server to allow a privileged APPC application to open an APPC conversation using the Single Sign-On feature on behalf of any defined Windows 2000 user. This is referred to as the privileged proxy feature. An extension has been added to the APPC **ALLOCATE** verb to invoke this feature.

An APPC application becomes privileged by being started in a Windows 2000 user account that is a member of a special Windows 2000 group. When a Host Security Domain is configured, SNA Manager will define a second Windows 2000 group for use with the host security features of Host Integration Server 2009. If the user account under which the actual client is running is a member of this second Windows 2000 group, the client is privileged to initiate an APPC conversation on behalf of any user account defined in the Host Account Cache.

The following illustrates how the privileged proxy feature works:

The Host Integration Server 2009 administrator creates a Host Security Domain called APP. SNA Manager now creates two

Windows 2000 groups. The first group is called APP and the second is called APP_PROXY for this example. Users that are assigned to the APP group are enabled for Single Sign-On. Users assigned to the APP_PROXY group are privileged proxies. The administrator adds the Windows 2000 user AppcUser to the APP_PROXY group using the Users button on the Host Security Domain property dialog box in SNA Manager.

The administrator then sets up an APPC application on the Host Integration Server to run as a Windows 2000 service called APPCAPP and that service has been set up to operate under the AppcUser user account. When APPCAPP runs, it opens an APPC session via an **ALLOCATE** verb using the extended VCB format and specifies the Windows 2000 user name of the desired user, UserA (for example).

The SNA Service sees the session request coming from a connection that is a member of the Host Security Domain APP. The Client/Server interface tells the SNA Service that the actual client is AppcUser.

The SNA Service checks to see if AppcUser is a member of the APP_PROXY group. Because AppcUser is a member of APP_PROXY, the SNA Service inserts the Username/Password for UserA in the APPC Attach (FMH-5) command and sends it off to the partner TP.

In order to support the privileged proxy feature, the APPC application must implement the following program logic:

The APPC application must determine the Windows 2000 user ID and domain name that it wishes to impersonate.

The APPC application must set the following parameters before calling the **ALLOCATE** verb:

Enable the use of the extended **ALLOCATE** verb control block structure by setting the AP_EXTD_VCB flag in the **opext** field.

Set security to AP_PROXY_SAME, AP_PROXY_PGM or AP_PROXY_STRONG.

Set up the pointers for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated.

Note
The application does not need to set up the **user_id** and **pwd** fields in the **ALLOCATE** VCB.

When the APPC application performs the above steps and issues the **ALLOCATE** verb, the Host Integration Server 2009 server will perform a lookup in the host security domain for the specified Windows 2000 user and set the user ID and password fields in the FMH-5 Attach message sent to the remote system.

CONFIRM

The **CONFIRM** verb sends the contents of the local logical unit's (LU) send buffer and a confirmation request to the partner transaction program (TP).

The following structure describes the verb control block used by the **CONFIRM** verb.

Syntax

```
struct confirm {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned long   conv_id;
    unsigned char   rts_rcvd;
};
```

Remarks

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_CONFIRM.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED.

conv_id

Returned parameter. Identifies the conversation established between the two TPs.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued REQUEST_TO_SEND, which requests the local TP to change the conversation to RECEIVE state.

To change to RECEIVE state operating on Microsoft Windows 2000 the local TP can use PREPARE_TO_RECEIVE, RECEIVE_AND_WAIT, or RECEIVE_AND_POST.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_CONFIRM_ON_SYNC_LEVEL_NONE

Secondary return code; the local TP attempted to use **CONFIRM** in a conversation with a synchronization level of AP_NONE. The synchronization level, established by ALLOCATE, must be AP_CONFIRM_SYNC_LEVEL.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_CONFIRM_BAD_STATE

Secondary return code; the conversation was not in SEND state.

AP_CONFIRM_NOT_LL_BDY

Secondary return code; the conversation for the local TP was in SEND state, and the local TP did not finish sending a logical record.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after **ALLOCATE**.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE, AP_CONFIRM_SYNC_LEVEL, or AP_SYNCPT) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer has encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued SEND_ERROR with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued DEALLOCATE with **dealloc_type**set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

Remarks

In response to **CONFIRM**, the partner TP normally issues CONFIRMED to confirm that it has received the data without error. (If the partner TP encounters an error, it issues SEND_ERROR or abnormally deallocates the conversation.)

The TP can issue **CONFIRM** only if the conversation's synchronization level, established by ALLOCATE, is AP_CONFIRM_SYNC_LEVEL.

The conversation must be in SEND state when the TP issues this verb. State changes, summarized in the following table, are based on the value of the **primary_rc**.

primary_rc	**New state**
AP_OK	No change
AP_ALLOCATION_ERROR	RESET
AP_COMM_SUBSYSTEM_ABENDED AP_COMM_SUBSYSTEM_NOT_LOADED	RESET RESET
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	RESET RESET
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	RESET RESET RESET RESET
AP_PROG_ERROR_PURGING AP_SVC_ERROR_PURGING	RECEIVE RECEIVE

CONFIRM waits for a response from the partner TP. A response is generated by one of the following verbs in the partner TP:

- CONFIRMED
- SEND_ERROR
- DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- TP_ENDED

By issuing **CONFIRM** afterALLOCATE, the invoking TP can immediately determine whether the allocation was successful (if **synclevel** is set to AP_CONFIRM_SYNC_LEVEL).

Normally, the value of the **ALLOCATE** verb's **mode_name** parameter must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU.

If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**. For more information, see Host Integration Server 2009 Help.

Several parameters of **ALLOCATE** are EBCDIC or ASCII strings. A TP can use the common service verb (CSV) CONVERT to translate a string from one character set to the other.

To send the **ALLOCATE** request immediately, the invoking TP can issue FLUSH or **CONFIRM** immediately after **ALLOCATE**. Otherwise, the **ALLOCATE** request accumulates with other data in the local LU's send buffer until the buffer is full.

CONFIRMED

The **CONFIRMED** verb responds to a confirmation request from the partner transaction program (TP). It informs the partner TP that the local TP has not detected an error in the received data. Because the TP issuing the confirmation request waits for a confirmation, **CONFIRMED** synchronizes the processing of the two TPs.

The following structure describes the verb control block (VCB) used by the **CONFIRMED** verb.

Syntax

```
struct confirmed {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned long   conv_id;
    unsigned char   rts_rcvd;
};
```

Remarks

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_CONFIRMED.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTEDin the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Identifies the conversation established between the two TPs. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued MC_REQUEST_TO_SEND, which requests the local TP to change the conversation to RECEIVE state.

To change to RECEIVE state the local TP can use MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, or MC_RECEIVE_AND_POST.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_CONFIRMED_BAD_STATE

Secondary return code; the conversation is not in CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation must be in one of the following states when the TP issues this verb:

- CONFIRM

- CONFIRM_SEND
- CONFIRM_DEALLOCATE

The new state is determined by the old state—the state of the conversation when the local TP issued **CONFIRMED**. The old state is indicated by the value of the **what_rcvd** parameter of the preceding receive verb. The following state changes are possible:

Old state	New state
CONFIRM	RECEIVE
CONFIRM_SEND	SEND
CONFIRM_DEALLOCATE	RESET

Confirmation Requests

A confirmation request is issued by one of the following verbs in the partner TP:

- CONFIRM
- PREPARE_TO_RECEIVE if **ptr_type** is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by ALLOCATE) is AP_CONFIRM_SYNC_LEVEL
- DEALLOCATE if **dealloc_type** is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by **ALLOCATE**) is AP_CONFIRM_SYNC_LEVEL
- SEND_DATA if type is set to AP_SEND_DATA_CONFIRM and the conversation's synchronization level (established by **ALLOCATE**) is AP_CONFIRM_SYNC_LEVEL

A confirmation request is received by the local TP through the **what_rcvd** parameter of one of the following verbs:

- RECEIVE_IMMEDIATE
- RECEIVE_AND_WAIT
- RECEIVE_AND_POST

CONFIRMED is issued by the local TP only if **what_rcvd** contains one of the following values:

- AP_CONFIRM_WHAT_RECEIVED
- AP_CONFIRM_SEND
- AP_CONFIRM_DEALLOCATE

If the **rtn_status** parameter is set to AP_YES, **what_rcvd** can also contain the following values:

- AP_DATA_COMPLETE_CONFIRM
- AP_DATA_COMPLETE_CONFIRM_SEND
- AP_DATA_COMPLETE_CONFIRM_DEALL

For basic conversations, **what_rcvd** can also contain the following values:

- AP_DATA_CONFIRM
- AP_DATA_CONFIRM_SEND
- AP_DATA_CONFIRM_DEALLOCATE

DEALLOCATE

The **DEALLOCATE** verb deallocates a conversation between two transaction programs (TPs).

The following structure describes the verb control block (VCB) used by the **DEALLOCATE** verb.

Syntax

```
struct deallocate {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        tp_id[8];
    unsigned long        conv_id;
    unsigned char        reserv3;
    unsigned char        dealloc_type;
    unsigned short       log_dlen;
    unsigned char FAR *  log_dptr;
    void                 (WINAPI *callback)();
    void                 *correlator;
    unsigned char        reserv6[4];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_DEALLOCATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Identifies the conversation established between the two TPs. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

dealloc_type

Supplied parameter. Specifies how to perform the deallocation.

Using one of the following values deallocates the conversation abnormally:

- AP_ABEND_PROG

- AP_ABEND_SVC
- AP_ABEND_TIMER

If the conversation is in SEND state when the local TP issues **DEALLOCATE**, APPC sends the contents of the local logical unit's (LU) send buffer to the partner TP before deallocating the conversation. If the conversation is in RECEIVE or PENDING_POST state, APPC purges any incoming data before deallocating the conversation.

An application or service TP should specify AP_ABEND_PROG when it encounters an error preventing the successful completion of a transaction.

A service TP should specify AP_ABEND_SVC when it encounters an error caused by its partner service TP (for example, a format error in control information sent by the partner service TP). A service TP should specify AP_ABEND_TIMER when it encounters an error requiring immediate deallocation (for example, an operator ending the program prematurely).

AP_FLUSH sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation. This value is allowed only if the conversation is in SEND state.

AP_SYNC_LEVEL uses the conversation's synchronization level (established by ALLOCATE) to determine how to deallocate the conversation. This value is allowed only if the conversation is in SEND state.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation.

If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC deallocates the conversation. If, however, the partner TP reports an error, the conversation remains allocated.

log_dlen

Supplied parameter. Specifies the number of bytes of data to be sent to the error log file. The range is from 0 through 32767.

You can set this parameter to a number greater than zero if **dealloc_type** is set to AP_ABEND_PGM, AP_ABEND_SVC, or AP_ABEND_TIMER. Otherwise, this parameter must be zero.

log_dptr

Supplied parameter. Provides the address of the data buffer containing error information. The data is sent to the local error log and to the partner LU.

This parameter is used by **DEALLOCATE** if **log_dlen** is greater than zero.

For Microsoft Windows 2000, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

For OS/2, the log data buffer must reside on an unnamed, shared segment, which is allocated by the function **DosAllocSeg** with Flags equal to 1. The log data buffer must fit entirely on the segment.

The TP must format the error data as a GDS error log variable. For more information, see your IBM SNA manual(s).

callback

Supplied parameter. Only present if the AP_EXTD_VCB bit is set in the **opext** member, indicating support for Sync Point. This parameter is the address of a user-supplied callback function. If this field is NULL, no notification will be provided.

The prototype of the callback routine is as follows:

```
void WINAPI callback_proc(
    struct appc_hdr *vcb,
    unsigned char tp_id[8],
    unsigned long conv_id,
    unsigned short type,
    void *correlator
   );
```

The callback procedure can take any name, since the address of the procedure is passed to the APPC DLL. The parameters passed to the function are as follows:

vcb

A pointer to the **DEALLOCATE** verb control block that caused the conversation to be deallocated.

tp_id

The TP identifier of the TP that owned the deallocated conversation.

conv_id

The conversation identifier of the deallocated conversation.

type

The type of the message flow that caused the callback to be invoked. Possible values are:

AP_DATA_FLOW

Normal data flow on the session.

AP_UNBIND

The session was unbound normally.

AP_FAILURE

The session terminated due to an outage.

correlator

This value is the **correlator** specified on the **DEALLOCATE** verb.

correlator

Supplied parameter. Only present if the AP_EXTD_VCB bit is set in the **opext** member, indicating support for the Sync Point API. This **correlator** field allows the TP to specify a value it can use to correlate a call to the callback function with, for example, its own internal data structures. This value is returned to the TP as one of the parameters of the callback routine when it is invoked.

reserv4

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_DEALLOC_BAD_TYPE

Secondary return code; the **dealloc_type** parameter was not set to a valid value.

AP_DEALLOC_LOG_LL_WRONG

Secondary return code; the LL field of the GDS error log variable did not match the actual length of the log data.

AP_INVALID_DATA_SEGMENT

Secondary return code; the error data for the log file was longer than the segment allocated to contain the error data, or the address of the error data buffer was wrong.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_DEALLOC_CONFIRM_BAD_STATE

Secondary return code; the conversation was not in SEND state, and the TP attempted to flush the send buffer and send a confirmation request. This attempt occurred because the value of **dealloc_type** was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_CONFIRM_SYNC_LEVEL.

AP_DEALLOC_FLUSH_BAD_STATE

Secondary return code; the conversation was not in SEND state and the TP attempted to flush the send buffer. This attempt occurred because the value of **dealloc_type** was AP_FLUSH or because the value of **dealloc_type** was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_NONE. In either case, the conversation must be in SEND state.

AP_DEALLOC_NOT_LL_BDY

Secondary return code; the conversation was in SEND state, and the TP did not finish sending a logical record. The **dealloc_type** parameter was set to AP_SYNC_LEVEL or AP_FLUSH.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued SEND_ERROR with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type**

set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

Remarks

Depending on the value of the **dealloc_type** parameter, the conversation can be in one of the states indicated in the following table when the TP issues **DEALLOCATE**.

dealloc_type	**Allowed state**
AP_FLUSH	SEND
AP_SYNC_LEVEL	SEND
AP_ABEND	Any except RESET
AP_ABEND_PROG	Any except RESET
AP_ABEND_SVC	Any except RESET
AP_ABEND_TIMER	Any except RESET

State changes, summarized in the following table, are based on the value of the **primary_rc**.

primary_rc	**New state**
AP_OK	RESET
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

Before deallocating the conversation, this verb performs the equivalent of one of the following:

- FLUSH, by sending the contents of the local LU's send buffer to the partner LU (and TP).
- CONFIRM, by sending the contents of the local LU's send buffer and a confirmation request to the partner TP.

After this verb has successfully executed, the conversation identifier is no longer valid.

LU 6.2 Sync Point can use an optimization of the message flows known as implied forget. When the protocol specifies that a FORGET PS header is required, the next data flow on the session implies that a FORGET has been received. In the normal situation, the TP is aware of the next data flow when data is received or sent on one of its Sync Point conversations.

However, it is possible that the last message to flow is caused by the conversation being deallocated. In this case, the TP is unaware when the next data flow on the session occurs. To provide the TP with this notification, the **DEALLOCATE** verb is modified to allow the TP to register a callback function which will be called:

- On the first normal flow transmission (request or response) over the session used by the conversation.
- If the session is unbound before any other data flows.
- If the session is terminated abnormally due to a DLC outage.

The **DEALLOCATE** verb also contains a **correlator** field member that is returned as one of the parameters when the callback function is invoked. The application can use this parameter in any way (for example, as a pointer to a control block within the application).

The TP can use the *type* parameter passed to the callback function to determine whether the message flow indicates an implied forget has been received.

Note that the **DEALLOCATE** verb will probably complete before the callback routine is called. The conversation is considered to be in RESET state and no further verbs can be issued using the conversation identifier. If the application issues a TP_ENDED verb before the next data flow on the session, the callback routine will not be invoked.

Host Integration Server 2009 allows TPs to deallocate conversations immediately after sending data by specifying the **type** parameter on SEND_DATA as AP_SEND_DATA_DEALLOC_*. However, the SEND_DATA verbs do not contain the implied forget callback function. TPs that want to receive implied forget notification must issue **DEALLOCATE** explicitly.

FLUSH

The **FLUSH** verb sends the contents of the local logical unit's (LU) send buffer to the partner LU and transaction program (TP). If the send buffer is empty, no action takes place.

The following structure describes the verb control block (VCB) used by the **FLUSH** verb.

Syntax

```
struct flush {
    unsigned short   opcode;
        unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_FLUSH.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_FLUSH_NOT_SEND_STATE

Secondary return code; the conversation was not in SEND state.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

Data processed by SEND_DATA accumulates in the local LU's send buffer until one of the following happens:

- The local TP issues **FLUSH** (or other verb that flushes the LU's send buffer).
- The buffer is full.

The request generated by ALLOCATE is also buffered.

The conversation must be in SEND state when the TP issues this verb.

There is no state change.

GET_ATTRIBUTES

The **GET_ATTRIBUTES** verb returns the attributes of the conversation.

The following structure describes the verb control block (VCB) used by the **GET_ATTRIBUTES** verb.

Syntax

```
struct get_attributes {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    reserv3;
    unsigned char    sync_level;
    unsigned char    mode_name[8];
    unsigned char    net_name[8];
    unsigned char    lu_name[8];
    unsigned char    lu_alias[8];
    unsigned char    plu_alias[8];
    unsigned char    plu_un_name[8];
    unsigned char    reserv4[2];
    unsigned char    fqplu_name[17];
    unsigned char    reserv5;
    unsigned char    user_id[10];
    unsigned long    conv_group_id;
    unsigned char    conv_corr_len;
    unsigned char    conv_corr[8];
    unsigned char    reserv6[13];
NOTE: The following fields are present when the high bit of opext is set (opext & AP_EXTD_V
CB) != 0.
    unsigned char    luw_id[26];
    unsigned char    sess_id[8];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_GET_ATTRIBUTES.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

sync_level

Returned parameter. Specifies the level of synchronization processing for the conversation. This parameter determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

- AP_NONE indicates that confirmation processing will not be used in this conversation.
- AP_CONFIRM_SYNC_LEVEL indicates that TPs can use confirmation processing in this conversation.
- AP_SYNCPT indicates that TPs can use Sync Point Level 2 confirmation processing in this conversation.

mode_name

Returned parameter. Specifies the name of a set of networking characteristics. It is a type A EBCDIC character string.

net_name

Returned parameter. Specifies the name of the SNA network containing the local logical unit (LU) used by this TP. It is a type A EBCDIC character string.

lu_name

Returned parameter. Provides the name of the local LU.

lu_alias

Returned parameter. Provides the alias by which the local LU is known to the local TP. It is an ASCII character string.

plu_alias

Returned parameter. Provides the alias by which the partner LU is known to the local TP. It is an ASCII character string.

plu_un_name

Returned parameter. Specifies the uninterpreted name of the partner LU—the name of the partner LU as defined to the system services control point (SSCP). It is a type AE EBCDIC character string. This parameter is returned only if the local LU is dependent.

fqplu_name

Returned parameter. Provides the fully qualified name of the partner LU. It is a type A EBCDIC character string. The field contains the network name, an EBCDIC period, and the partner-LU name.

user_id

Returned parameter. Specifies the user identifier sent by the invoking TP through ALLOCATE to access the invoked TP (if applicable). It is a type AE EBCDIC character string. The field contains the user identifier if the following conditions are true:

- The invoked TP requires conversation security.
- **GET_ATTRIBUTES** was issued by the invoked TP.

Otherwise, the field contains spaces.

conv_group_id

Returned parameter. Specifies the conversation group identifier for the session to which the conversation has been allocated. This is also returned on ALLOCATE and RECEIVE_ALLOCATE.

conv_corr_len

Returned parameter. Specifies the length of the conversation correlator identifier that is returned.

conv_corr

Returned parameter. Specifies the conversation correlator identifier (if any) that the source LU assigns to identify the conversation, which is unique for the source/partner LU pair. It is sent by the source LU on the allocation request.

Note
The following fields are present when the high bit of **opext** is set(**opext** & AP_EXTD_VCB) != 0.These fields are only present when using Sync Point Level 2 support.

luw_id

Logical unit-of-work identifier.

sess_id

Session identifier.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

GET_LU_STATUS

The **GET_LU_STATUS** verb returns the status of a particular logical unit (LU). This conversation verb is only available when Sync Point conversations are supported.

The following structure describes the verb control block (VCB) used by the **GET_LU_STATUS** verb.

Syntax

```
struct get_type {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned char    plu_alias[8];
    unsigned short   active_sess;
    unsigned char    zero_sess;
    unsigned char    local_only;
    unsigned char    synchpoint;
    unsigned char    pool_member;
    unsigned char    reserv3[7];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_GET_LU_STATUS.

opext

This field is unused by the **GET_LU_STATUS** verb.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter was returned by TP_STARTED in the invoking TP, or by RECEIVE_ALLOCATE or RECEIVE_ALLOCATE_EX in the invoked TP.

plu_alias

Supplied parameter. Provides the identifier for the LU about which this TP is inquiring. The value of this parameter was returned by MC_ALLOCATE or ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

Not required if local_only is set to AP_YES

active_sess

Returned parameter. Supplies the number of active sessions on this LU.

zero_sess

Returned parameter. Indicates whether a zero session is on this LU. Values are AP_YES or AP_NO.

active_sess

Returned parameter.

zero_sess

Returned parameter.

local_only

If this field is set to AP_YES then the plu_alias does not need to be specified and the verb only returns the local LU information - syncpoint and default_pool.

synchpoint

Returned parameter.

pool_member

If this field is set to AP_YES then the plu_alias does not need to be specified and the verb only returns the local LU information - syncpoint and default_pool.

reserv3

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_LU_ALIAS

Secondary return code; the value of **plu_alias** did not match any LUs assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

The current version of GET_LU_STATUS allows for an application to retrieve configuration parameters for a Local APPC LU.

To check the configuration of a particular Local LU before issuing a RECEIVE_ALLOCATE_EX verb, the following verb sequence should be issued:

- TP_STARTED (specifying the Local LU of interest)
- GET_LU_STATUS (with local_only set to AP_YES)
- TP_ENDED (AP_SOFT)

GET_STATE

The **GET_STATE** verb returns the state of a particular conversation.

The following structure describes the verb control block (VCB) used by the **GET_STATE** verb.

Syntax

```
struct get_state {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    conv_state;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_GET_STATE.

opext

This field is unused by the **GET_STATE** verb.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the identifier for the conversation about which this TP is inquiring. The value of this parameter was returned by MC_ALLOCATE or ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

conv_state

Returned parameter. Indicates the state of the conversation. The **conv_state** parameter can be one of the following values:

AP_RESET_STATE

The conversation is in the RESET state.

AP_SEND_STATE

The conversation is in the SEND state.

AP_RECEIVE_STATE

The conversation is in the RECEIVE state.

AP_CONFIRM_STATE

The conversation is in the CONFIRM state.

AP_CONFIRM_SEND_STATE

The conversation is in the CONFIRM_SEND state.

AP_CONFIRM_DEALL_STATE

The conversation is in the CONFIRM_DEALLOCATE state.

AP_PEND_POST_STATE

The conversation has a **POST** verb pending.

AP_PEND_DEALL_STATE

The conversation has a **DEALLOCATE** verb pending.

AP_END_CONV_STATE

The conversation is in the END_CONVERSATION state.

AP_SEND_PENDING_STATE

The conversation is in the SEND_PENDING state.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state when the TP issues this verb.

There is no state change.

GET_TYPE

The **GET_TYPE** verb returns the conversation type (basic or mapped) of a particular conversation so the transaction program (TP) can decide whether to use basic or mapped conversation verbs.

The following structure describes the verb control block (VCB) used by the **GET_TYPE** verb.

Syntax

```
struct get_type {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    conv_type;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_GET_TYPE.

opext

This field is unused by the **GET_TYPE** verb.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the identifier for the conversation about which this TP is inquiring. The value of this parameter was returned by MC_ALLOCATE or ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

conv_type

Returned parameter. Supplies the type of conversation, either AP_BASIC_CONVERSATION or AP_MAPPED_CONVERSATION.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

MC_ALLOCATE

The **MC_ALLOCATE** verb is issued by the invoking transaction program (TP). It allocates a session between the local logical unit (LU) and partner LU and (in conjunction with RECEIVE_ALLOCATE) establishes a conversation between the invoking TP and the invoked TP. After this verb executes successfully, APPC generates a conversation identifier (**conv_id**). The **conv_id** is a required parameter for all other APPC conversation verbs.

The following structure describes the verb control block (VCB) used by the **MC_ALLOCATE** verb.

Syntax

```
struct mc_allocate {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    reserv3;
    unsigned char    synclevel;
    unsigned char    reserv4[2];
    unsigned char    rtn_ctl;
    unsigned char    reserv5;
    unsigned long    conv_group_id;
    unsigned long    sense_data;
    unsigned char    plu_alias[8];
    unsigned char    mode_name[8];
    unsigned char    tp_name[64];
    unsigned char    security;
    unsigned char    reserv6[11];
    unsigned char    pwd[10];
    unsigned char    user_id[10];
    unsigned short   pip_dlen;
    unsigned char FAR * pip_dptr;
    unsigned char    reserv7;
    unsigned char    fqplu_name[17];
    unsigned char    reserv8[8];
    unsigned long    proxy_user;
    unsigned long    proxy_domain;
    unsigned char    reserv9[16];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code; AP_M_ALLOCATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION. If the AP_EXTD_VCB bit is set, this indicates that an extended version of the verb control block is used. In this case, the **MC_ALLOCATE** structure includes Sync Point support or privileged proxy feature support.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED.

conv_id

Returned parameter. Identifies the conversation established between the two TPs.

reserv3

A reserved field.

synclevel

Supplied parameter. Specifies the synchronization level of the conversation. It determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

- AP_NONE specifies that confirmation processing will not be used in this conversation.
- AP_CONFIRM_SYNC_LEVEL specifies that the TPs can use confirmation processing in this conversation.
- AP_SYNCPT specifies that TPs can use Sync Point Level 2 confirmation processing in this conversation.

reserv4

A reserved field.

reserv5

A reserved field.

rtn_ctl

Supplied parameter. Specifies when the local LU, acting on a session request from the local TP, should return control to the local TP. For information about sessions, see Transaction Programs Overview.

- AP_IMMEDIATE specifies that the LU allocates a contention-winner session, if one is immediately available, and returns control to the TP.
- AP_WHEN_SESSION_ALLOCATED specifies that the LU does not return control to the TP until it allocates a session or encounters one of the errors documented in Return Codes in this topic. (If the session limit is zero, the LU returns control immediately.) If a session is not available, the TP waits for one.
- AP_WHEN_SESSION_FREE specifies that the LU allocates a contention-winner or contention-loser session, if one is available or able to be activated, and returns control to the TP. If an error occurs, (as documented in Return Codes in this topic) the call will return immediately with the error in the **primary_rc** and **secondary_rc** fields.
- AP_WHEN_CONWINNER_ALLOC specifies that the LU does not return control until it allocates a contention-winner session or encounters one of the errors documented in Return Codes in this topic. (If the session limit is zero, the LU returns control immediately.) If a session is not available, the TP waits for one.
- AP_WHEN_CONV_GROUP_ALLOC specifies that the LU does not return control to the TP until it allocates the session specified by **conv_group_id**or encounters one of the errors documented in Return Codes in this topic. If a session is not available, the TP waits for it to become free.

Note
AP_IMMEDIATE is the only value for **rtn_ctl** that never causes a new session to start. For values other than AP_IMMEDIATE, if an appropriate session is not immediately available, Microsoft® Host Integration Server tries to start one. This causes the on-demand connection to be activated.

conv_group_id

Supplied/returned parameter. Specifies the identifier of the conversation group from which the session should be allocated.

The **conv_group_id** is required only if **rtn_ctl**is set to WHEN_CONV_GROUP_ALLOC. When**rtn_ctl** specifies a different value and the **primary_rc** is AP_OK, this is a returned value.

sense_data

Returned parameter. Indicates an allocation error (retry or no-retry) and contains sense data.

plu_alias

Supplied parameter. Specifies the alias by which the partner LU is known to the local TP.

The **plu_alias** must match the name of a partner LU established during configuration.

The parameter is an 8-byte ASCII character string. It can consist of the following ASCII characters:

- Uppercase letters
- Numerals 0 through 9
- Spaces
- Special characters $, #, %, and @

The first character of this string cannot be a space.

If the value of this parameter is fewer than eight bytes, pad it on the right with ASCII spaces (0x20).

If you want to specify the partner LU with the **fqplu_name** parameter, fill this parameter with binary zeros.

For a user or group using TPs, 5250 emulators, and/or APPC applications, the system administrator can assign default local and remote LUs. In this case, the field is left blank or null and the default LUs are accessed when the user or group member starts an APPC program. For more information on default LUs, see Host Integration Server 2009 Help.

mode_name

Supplied parameter. Specifies the name of a set of networking characteristics defined during configuration.

The value of **mode_name** must match the name of a mode associated with the partner LU during configuration.

The parameter is an 8-byte EBCDIC character string. It can consist of characters from the type A EBCDIC character set:

- Uppercase letters
- Numerals 0 through 9
- Special characters $, #, and @

The first character in the string must be an uppercase letter or a special character.

Do not use SNASVCMG in a mapped conversation. SNASVCMG is a reserved **mode_name** used internally by APPC.

tp_name

Supplied parameter. Specifies the name of the invoked TP. The value of **tp_name** specified by **MC_ALLOCATE** in the invoking TP must match the value of **tp_name** specified by RECEIVE_ALLOCATE in the invoked TP.

The parameter is a 64-byte EBCDIC character string and is case-sensitive. The **tp_name** parameter can consist of the following EBCDIC characters:

- Uppercase and lowercase letters
- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If **tp_name** is fewer than 64 bytes, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention is that a service TP name can have up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the type AE EBCDIC character set.

security

Supplied parameter. Provides the information that the partner LU requires to validate access to the invoked TP. See the section **Possible values for the Security parameter** in this topic.

Reserv6

A reserved field.

pwd

Supplied parameter. Specifies the password associated with **user_id**.

The **pwd** parameter is required only if **security** is set to AP_PGM or AP_SAME. It must match the password for **user_id** that was established during configuration.

The **pwd** parameter is a 10-byte EBCDIC character string and is case-sensitive. It can consist of the following EBCDIC characters:

- Uppercase and lowercase letters
- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

If the APPC automatic logon feature is to be used, the **pwd** character string must be hard-coded to MS$SAME. See the Remarks section for details.

user_id

Supplied parameter. Specifies the user identifier required to access the partner TP. It is required only if the security parameter is set to AP_PGM or AP_SAME.

The **user_id** parameter is a 10-byte EBCDIC character string and is case-sensitive. It must match one of the user identifiers configured for the partner TP.

The parameter can consist of the following EBCDIC characters:

- Uppercase and lowercase letters
- Numerals 0 through 9
- Special characters $, #, @, and period (.)

If **user_id** is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

If the APPC automatic logon feature is to be used, the **user_id** character string must be hard-coded to MS$SAME. See the Remarks section for details.

pip_dlen

Supplied parameter. Specifies the length of the program initialization parameters (PIP) to be passed to the partner TP. The range is from 0 through 32767.

pip_dptr

Supplied parameter. Specifies the address of the buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

PIP data can consist of initialization parameters or environmental setup information required by a partner TP or remote operating system. The PIP data must follow the general data stream (GDS) format. For more information, see your IBM SNA manual(s).

For Microsoft Windows 2000, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

reserv7

A reserved field.

fqplu_name

Supplied parameter. Specifies the fully qualified name of the partner LU. This must match the fully qualified name of the local LU defined in the remote node. The parameter consists of two type A EBCDIC character strings for the NETID and the LU name of the partner LU. The names are separated by an EBCDIC period (.).

This name must be provided if no **plu_alias** is specified. It can consist of the following EBCDIC characters:

- 18Uppercase letters
- Numerals 0 through 9
- Special characters $, #, and @

If the value of this parameter is fewer than 17 bytes, pad it on the right with EBCDIC spaces (0x40).

reserv8

A reserved field.

proxy_user

Supplied parameter. Specifies a LPWSTR pointing to a Unicode string containing the user name to be impersonated using the privileged proxy feature. This field can only be used when the AP_EXTD_VCB bit is set on the **opext** field, indicating an extended VCB.

proxy_domain

Supplied parameter. Specifies a LPWSTR pointing to a Unicode string containing the domain name of the user to be impersonated using the privileged proxy feature. This field can only be used when the AP_EXTD_VCB bit is set on the **opext** field, indicating an extended VCB.

reserv9

A reserved field.

Possible Values for the Security Parameter

Based on the conversation security established for the invoked TP during configuration, use one of the following values:

- AP_NONE for an invoked TP that uses no conversation security.
- AP_PGM for an invoked TP that uses conversation security and thus requires a user identifier and password. Supply this information through the **user_id** and **pwd** parameters.
- AP_PROXY_PGM for an invoked TP with privileged proxy that uses conversation security and thus requires a user identifier and password. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields.
- AP_PROXY_SAME for a TP that has been invoked using privileged proxy with a valid user identifier and password supplied by the proxy, which in turn invokes another TP. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields.

  For example, assume that TP A invokes TP B with a valid user identifier and password supplied by the privileged proxy, and TP B in turn invokes TP C. If TP B specifies the value AP_PROXY_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator tells TP C to not require the password (if TP C is configured to accept an already-verified indicator).

- AP_PROXY_STRONG for an invoked TP with privileged proxy that uses conversation security and thus requires a user identifier and password provided by the privileged proxy mechanism. Pointers must be set up for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated. The application does not need to set the **user_id** and **pwd** fields. AP_PROXY_STRONG differs from AP_PROXY_PGM in that AP_PROXY_STRONG does not allow clear-text passwords. If the remote system does not support encrypted passwords (strong conversation security), then this call fails.

- AP_SAME for a TP that has been invoked with a valid user identifier and password, which in turn invokes another TP.

  For example, assume that TP A invokes TP B with a valid user identifier and password, and TP B in turn invokes TP C. If TP B specifies the value AP_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator tells TP C to not require the password (if TP C is configured to accept an already-verified indicator).

  When AP_SAME is used in an **MC_ALLOCATE** verb, your application must always provide values for the **user_id** and **pwd** parameters in the verb control block. Depending on the properties negotiated between Host Integration Server 2009 and the peer LU, the **MC_ALLOCATE** verb will send one of three kinds of Attach (FMH-5) messages, in this order of precedence:

  1. If the LUs have negotiated "already verified" security, then the Attach sent by Host Integration Server 2009 will not include the contents of the **pwd** parameter field specified in the VCB.

  2. If the LUs have negotiated "persistent verification" security, then the Attach sent by Host Integration Server will include the **pwd** parameter specified in the VCB, but only when the Attach is the first for the specified **user_id** parameter since the start of the LU-LU session, and will omit the **pwd** parameter on all subsequent Attaches (issued by your application or any other application using this LU-LU-mode triplet).

  3. If the LUs have not negotiated either of the above, then the Attach sent by Host Integration Server will omit both the **user_id** and **pwd** parameters on all Attaches.

     Your application cannot tell which mode of security has been negotiated between the LUs, nor can it tell whether the **MC_ALLOCATE** verb it is issuing is the first for that LU-LU-mode triplet. So your application must always set the **user_id** and **pwd** parameter fields in the VCB when **security** is set to AP_SAME.

     For more information on persistent verification and already verified security, see the SNA Formats Guide, section "FM Header 5: Attach (LU 6.2)".

- AP_STRONG for an invoked TP that uses conversation security and thus requires a user identifier and password. Supply this information through the **user_id** and **pwd** parameters. AP_STRONG differs from AP_PGM in that AP_STRONG does not allow clear-text passwords. If the remote system does not support encrypted passwords (strong conversation security), then this call fails.

  If the APPC automatic logon feature is to be used, **security** must be set to AP_PGM. See the Remarks section for details.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; the supplied parameter **rtn_ctl** specified immediate (AP_IMMEDIATE) return of control to the TP, and the local LU did not have an available contention-winner session.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_RETURN_CONTROL

Secondary return code; the value specified for **rtn_ctl** was not valid.

AP_BAD_SECURITY

Secondary return code; the value specified for **security** was not valid.

AP_BAD_SYNC_LEVEL

Secondary return code; the value specified for **sync_level** was not valid.

AP_BAD_TP_ID

Secondary return code; the value specified for **tp_id** was not valid.

AP_PIP_LEN_INCORRECT

Secondary return code; the value of **pip_dlen** was greater than 32767.

AP_UNKNOWN_PARTNER_MODE

Secondary return code; the value specified for **mode_name** was not valid.

AP_BAD_PARTNER_LU_ALIAS

Secondary return code; APPC did not recognize the supplied **partner_lu_alias**.

AP_NO_USE_OF_SNASVCMG

Secondary return code; SNASVCMG is not a valid value for **mode_name**.

AP_INVALID_DATA_SEGMENT

Secondary return code; the PIP data was longer than the allocated data segment, or the address of the PIP data buffer was wrong.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after **MC_ALLOCATE**.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **MC_ALLOCATE**, it can indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can

satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Host Integration Server system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

MC_ALLOCATE establishes a mapped conversation.

The conversation state is RESET when the TP issues this verb. After successful execution (**primary_rc** is AP_OK), the state changes to SEND. If the verb does not execute, the state remains unchanged.

Several parameters of **MC_ALLOCATE** are EBCDIC or ASCII strings. A TP can use the CSV CONVERT to translate a string from one character set to the other.

To send the **MC_ALLOCATE** request immediately, the invoking TP can issue MC_FLUSH or MC_CONFIRM immediately after **MC_ALLOCATE**. Otherwise, the **MC_ALLOCATE** request accumulates with other data in the local LU's send buffer until the buffer is full.

By issuing **MC_CONFIRM** after **MC_ALLOCATE**, the invoking TP can immediately determine whether the allocation was successful (if **synclevel** is set to AP_CONFIRM_SYNC_LEVEL).

Normally, the value of the **MC_ALLOCATE** verb's **mode_name** parameter must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU.

If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**.

Host Integration Server 2009 supports a feature called password substitution. This is a security feature supported by the latest version of the OS/400 operating system (V3R1) that encrypts any password that flows between two nodes on an Attach message. A password flows on an Attach whenever someone invokes an APPC transaction program specifying a user identifier and password. For example, this happens whenever anyone logs on to an AS/400.

Support for password substitution is indicated by setting bit 5 in byte 23 of the BIND request to 1 (which indicates that password substitution is supported). If the remote system sets this bit in the BIND response, Host Integration Server automatically encrypts the LU 6.2 conversation security password included in the FMH-5 Attach message. Host Integration Server APPC applications automatically take advantage of this feature by setting the **security** field of the VCB to AP_PGM or AP_STRONG in the **MC_ALLOCATE** request.

If an APPC application wants to force an encrypted password to flow, the application can specify AP_STRONG for the **security** field in the VCB in the **MC_ALLOCATE** request. This option is implemented as defined in OS/400 V3R1, and is documented in the OS/400 CPI-C programmer reference as CM_SECURITY_PROGRAM_STRONG, where the LU 6.2 **pwd** (password) field is encrypted before it flows over the physical network.

The password substitution feature is currently only supported by OS/400 V3R1 or later. If the remote system does not support this feature, Host Integration Server will UNBIND the session with the sense code of 10060006. The two nodes negotiate whether or not they support this feature in the BIND exchange. Host Integration Server sets a bit in the BIND, and also adds some random data on the BIND for encryption. If the remote node supports password substitution, it sets the same bit in the BIND response, and adds some (different) random data for decryption.

Host Integration Server 2009 supports automatic logon for APPC applications. This feature requires specific configuration by the network administrator: The APPC application must be invoked on the LAN side from a client of Host Integration Server. The client must be logged into a Windows 2000 domain, but can be any platform that supports the Host Integration Server 2009 APPC APIs.

The client application is coded to use "program" level security, with a special hard-coded APPC user name MS$SAME and password MS$SAME. When this session allocation flows from client to Host Integration Server, the Host Integration Server server looks up the host account and password corresponding to the Windows 2000 account under which the client is logged in, and substitutes the host account information into the APPC attach message it sends to the host.

Note
It is illegal for the remote node to set the bit specifying password substitution and not add the random data.

According to IBM, there are implementations of LU 6.2 password substitution that do not support password substitution but do echo the password substitution bit back to Host Integration Server, without specifying any random data. When they do this, Host Integration Server will UNBIND the session with the sense code 10060006.This sense code is interpreted as:

- 1006 = Required field or parameter missing.
- 0006 = A required subfield of a control vector was omitted.

Host Integration Server should also log an Event 17 (APPC session activation failure: BIND negative response sent).

The correct solution is for the failing implementation to be fixed. However, as a short-term workaround, the following Host Integration Server SNA Service registry setting can be set:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\snaservr\parameters\NOPWDSUB: REG_SZ: YES

When this parameter is specified in the registry, Host Integration Server password substitution support will be disabled.

Several updates have been made to Host Integration Server to allow a privileged APPC application to open an APPC conversation using the Single Sign-On feature on behalf of any defined Windows 2000 user. This is referred to as the privileged proxy feature. An extension has been added to the APPC **MC_ALLOCATE** verb to invoke this feature.

An APPC application becomes privileged by being started in a Windows 2000 user account that is a member of a special Windows 2000 group. When a Host Security Domain is configured, SNA Manager will define a second Windows 2000 group for use with the host security features of Host Integration Server. If the user account under which the actual client is running is a member of this second Windows 2000 group, the client is privileged to initiate an APPC conversation on behalf of any user account defined in the Host Account Cache.

The following illustrates how the privileged proxy feature works:

The Host Integration Server administrator creates a Host Security Domain called APP. SNA Manager now creates two Windows 2000 groups. The first group is called APP and the second is called APP_PROXY for this example. Users that are assigned to the APP group are enabled for Single Sign-On. Users assigned to the APP_PROXY group are privileged proxies. The administrator adds the Windows 2000 user AppcUser to the APP_PROXY group using the Users button on the Host Security Domain property dialog box in SNA Manager.

The administrator then sets up an APPC application on the Host Integration Server server to run as a Windows 2000 service called APPCAPP, and that service has been set up to operate under the AppcUser user account. When APPCAPP runs, it opens an APPC session via an **ALLOCATE** verb using the extended VCB format and specifies the Windows 2000 user name of the desired user, UserA (for example).

The SNA Service sees the session request coming from a connection that is a member of the Host Security Domain APP. The Client/Server interface tells the SNA Service that the actual client is AppcUser.

The SNA Service checks to see if AppcUser is a member of the APP_PROXY group. Because AppcUser is a member of APP_PROXY, the SNA Service inserts the Username/Password for UserA in the APPC Attach (FMH-5) command and sends it off to the partner TP.

In order to support the privileged proxy feature, the APPC application must implement the following program logic:

The APPC application must determine the Windows 2000 user ID and domain name that it wishes to impersonate.

The APPC application must set the following parameters before calling the **MC_ALLOCATE** verb:

Enable the use of the extended **MC_ALLOCATE** verb control block structure by setting the AP_EXTD_VCB flag in the **opext** field.

Set security to AP_PROXY_SAME, AP_PROXY_PGM, or AP_PROXY_STRONG.

Set up the pointers for **proxy_user** and **proxy_domain** to point to Unicode strings containing the user name and domain name of the user to be impersonated.

Note
The application does not need to set up the **user_id** and **pwd** fields in the **MC_ALLOCATE** VCB.

When the APPC application performs the above steps and issues the **MC_ALLOCATE** verb, the Host Integration Server server will perform a lookup in the host security domain for the specified Windows 2000 user and set the user ID and password fields in the FMH-5 Attach message sent to the remote system.

MC_CONFIRM

The **MC_CONFIRM** verb sends the contents of the local logical unit's (LU) send buffer and a confirmation request to the partner transaction program (TP).

The following structure describes the verb control block (VCB) used by the **MC_CONFIRM** verb.

Syntax

```
struct mc_confirm {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned long   conv_id;
    unsigned char   rts_rcvd;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_CONFIRM.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED.

conv_id

Returned parameter. Identifies the conversation established between the two TPs.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued MC_REQUEST_TO_SEND, which requests the local TP to change the conversation to RECEIVE state.

To change to RECEIVE state the local TP can use MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, or MC_RECEIVE_AND_POST.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_CONFIRM_ON_SYNC_LEVEL_NONE

Secondary return code; the local TP attempted to use **MC_CONFIRM** in a conversation with a synchronization level of AP_NONE. The synchronization level, established by MC_ALLOCATE, must be AP_CONFIRM_SYNC_LEVEL.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_CONFIRM_BAD_STATE

Secondary return code; the conversation was not in SEND state.

AP_CONFIRM_NOT_LL_BDY

Secondary return code; the conversation for the local TP was in SEND state, and the local TP did not finish sending a logical record.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after **MC_ALLOCATE**.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE, AP_CONFIRM_SYNC_LEVEL, or AP_SYNCPT) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer has encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued MC_SEND_ERROR. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with**dealloc_type**set to AP_ABEND.

- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

In response to **MC_CONFIRM**, the partner TP normally issues MC_CONFIRMED to confirm that it has received the data without error. (If the partner TP encounters an error, it issues MC_SEND_ERROR or abnormally deallocates the conversation.)

The TP can issue **MC_CONFIRM** only if the conversation's synchronization level, established by MC_ALLOCATE, is AP_CONFIRM_SYNC_LEVEL.

The conversation must be in SEND state when the TP issues this verb. State changes, summarized in the following table, are based on the value of the **primary_rc**.

primary_rc	**New state**
AP_OK	No change
AP_ALLOCATION_ERROR	RESET
AP_COMM_SUBSYSTEM_ABENDED AP_COMM_SUBSYSTEM_NOT_LOADED	RESET RESET
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	RESET RESET
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	RESET RESET RESET RESET
AP_PROG_ERROR_PURGING AP_SVC_ERROR_PURGING	RECEIVE RECEIVE

MC_CONFIRM waits for a response from the partner TP. A response is generated by one of the following verbs in the partner TP:

- MC_CONFIRMED
- MC_SEND_ERROR
- MC_DEALLOCATE with **dealloc_type** set to AP_ABEND
- TP_ENDED

By issuing **MC_CONFIRM** afterMC_ALLOCATE, the invoking TP can immediately determine whether the allocation was successful (if **synclevel** is set to AP_CONFIRM_SYNC_LEVEL).

Normally, the value of the **MC_ALLOCATE** verb's **mode_name** parameter must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU.

If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**. For more information, see Host Integration Server 2009 Help.

Several parameters of **MC_ALLOCATE** are EBCDIC or ASCII strings. A TP can use the common service verb (CSV) CONVERT to translate a string from one character set to the other.

To send the **MC_ALLOCATE** request immediately, the invoking TP can issue MC_FLUSH or **MC_CONFIRM** immediately after **MC_ALLOCATE**. Otherwise, the **MC_ALLOCATE** request accumulates with other data in the local LU's send buffer until the buffer is full.

MC_CONFIRMED

The **MC_CONFIRMED** verb responds to a confirmation request from the partner transaction program (TP). It informs the partner TP that the local TP has not detected an error in the received data. Because the TP issuing the confirmation request waits for a confirmation, **MC_CONFIRMED** synchronizes the processing of the two TPs.

The following structure describes the verb control block (VCB) used by the **MC_CONFIRMED** verb.

Syntax

```
struct mc_confirmed {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned long   conv_id;
    unsigned char   rts_rcvd;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_CONFIRMED.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTEDin the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Identifies the conversation established between the two TPs. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued MC_REQUEST_TO_SEND, which requests the local TP to change the conversation to RECEIVE state.

To change to RECEIVE state the local TP can use MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, or MC_RECEIVE_AND_POST.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_CONFIRMED_BAD_STATE

Secondary return code; the conversation is not in CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation must be in one of the following states when the TP issues this verb:

- CONFIRM
- CONFIRM_SEND
- CONFIRM_DEALLOCATE

The new state is determined by the old state—the state of the conversation when the local TP issued **MC_CONFIRMED**. The old state is indicated by the value of the **what_rcvd** parameter of the preceding receive verb. The following state changes are possible:

Old state	New state
CONFIRM	RECEIVE
CONFIRM_SEND	SEND
CONFIRM_DEALLOCATE	RESET

Confirmation Requests

A confirmation request is issued by one of the following verbs in the partner TP:

- MC_CONFIRM
- MC_PREPARE_TO_RECEIVE if **ptr_type** is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by MC_ALLOCATE) is AP_CONFIRM_SYNC_LEVEL
- MC_DEALLOCATE if **dealloc_type** is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by MC_ALLOCATE) is AP_CONFIRM_SYNC_LEVEL
- MC_SEND_DATA if type is set to AP_SEND_DATA_CONFIRM and the conversation's synchronization level (established by MC_ALLOCATE) is AP_CONFIRM_SYNC_LEVEL

A confirmation request is received by the local TP through the **what_rcvd** parameter of one of the following verbs:

- MC_RECEIVE_IMMEDIATE
- MC_RECEIVE_AND_WAIT
- MC_RECEIVE_AND_POST

MC_CONFIRMED is issued by the local TP only if **what_rcvd** contains one of the following values:

- AP_CONFIRM_WHAT_RECEIVED
- AP_CONFIRM_SEND
- AP_CONFIRM_DEALLOCATE

If the **rtn_status** parameter is set to AP_YES, **what_rcvd** can also contain the following values:

- AP_DATA_COMPLETE_CONFIRM
- AP_DATA_COMPLETE_CONFIRM_SEND
- AP_DATA_COMPLETE_CONFIRM_DEALL

MC_DEALLOCATE

The **MC_DEALLOCATE** verb deallocates a conversation between two transaction programs (TP).

The following structure describes the verb control block (VCB) used by the **MC_DEALLOCATE** verb.

Syntax

```
struct mc_deallocate {
    unsigned short   opcode;
    unsigned char    opext;
     unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    reserv3;
    unsigned char    dealloc_type;
    unsigned char    reserv4[2];
    unsigned char    reserv5[4];
    void             (WINAPI *callback)();
    void             *correlator;
    unsigned char    reserv6[4];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_DEALLOCATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Identifies the conversation established between the two TPs. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

dealloc_type

Supplied parameter. Specifies how to perform the deallocation.

For **MC_DEALLOCATE**, use AP_ABEND to deallocate the conversation abnormally. If the conversation is in SEND state when the local TP issues **MC_DEALLOCATE**, APPC sends the contents of the local logical unit's (LU) send buffer to the partner TP before deallocating the conversation. If the conversation is in RECEIVE or PENDING_POST state, APPC purges any incoming

data before deallocating the conversation.

A TP should specify AP_ABEND when it encounters an error preventing the successful completion of a transaction.

AP_FLUSH sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation. This value is allowed only if the conversation is in SEND state.

AP_SYNC_LEVEL uses the conversation's synchronization level (established by MC_ALLOCATE) to determine how to deallocate the conversation. This value is allowed only if the conversation is in SEND state.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation.

If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC deallocates the conversation. If, however, the partner TP reports an error, the conversation remains allocated.

callback

Supplied parameter. Only present if the AP_EXTD_VCB bit is set in the **opext** member indicating support for Sync Point. This parameter is the address of a user-supplied callback function. If this field is NULL, no notification will be provided.

The prototype of the callback routine is as follows:

```
void WINAPI callback_proc(
    struct appc_hdr *vcb,
    unsigned char tp_id[8],
    unsigned long conv_id,
    unsigned short type,
    void *correlator
   );
```

The callback procedure can take any name, since the address of the procedure is passed to the APPC DLL. The parameters passed to the function are as follows:

vcb

A pointer to the **MC_DEALLOCATE** verb control block that caused the conversation to be deallocated.

tp_id

The TP identifier of the TP that owned the deallocated conversation.

conv_id

The conversation identifier of the deallocated conversation.

type

The type of the message flow that caused the callback to be invoked. Possible values are:

AP_DATA_FLOW

Normal data flow on the session.

AP_UNBIND

The session was unbound normally.

AP_FAILURE

The session terminated due to an outage.

correlator

This value is the **correlator** specified on the **MC_DEALLOCATE** verb.

correlator

Supplied parameter. Only present if the AP_EXTD_VCB bit is set in the **opext** member indicating support for the Sync Point API. This **correlator** field allows the TP to specify a value it can use to correlate a call to the callback function with, for example, its own internal data structures. This value is returned to the TP as one of the parameters of the callback routine when it is invoked.

reserv4

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_DEALLOC_BAD_TYPE

Secondary return code; the **dealloc_type** parameter was not set to a valid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_DEALLOC_CONFIRM_BAD_STATE

Secondary return code; the conversation was not in SEND state, and the TP attempted to flush the send buffer and send a confirmation request. This attempt occurred because the value of **dealloc_type** was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_CONFIRM_SYNC_LEVEL.

AP_DEALLOC_FLUSH_BAD_STATE

Secondary return code; the conversation was not in SEND state and the TP attempted to flush the send buffer. This attempt occurred because the value of **dealloc_type** was AP_FLUSH or because the value of **dealloc_type** was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_NONE. In either case, the conversation must be in SEND state.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued MC_SEND_ERROR. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued **MC_DEALLOCATE** with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

Depending on the value of the **dealloc_type** parameter, the conversation can be in one of the states indicated in the following table when the TP issues **MC_DEALLOCATE**.

Dealloc_type	**Allowed state**
AP_FLUSH	SEND
AP_SYNC_LEVEL	SEND
AP_ABEND	Any state except RESET
AP_ABEND_PROG	Any state except RESET
AP_ABEND_SVC	Any state except RESET
AP_ABEND_TIMER	Any state except RESET

State changes, summarized in the following table, are based on the value of the **primary_rc**.

Primary_rc	**New state**
AP_OK	RESET
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE

Before deallocating the conversation, this verb performs the equivalent of one of the following:

- MC_FLUSH, by sending the contents of the local LU's send buffer to the partner LU (and TP).
- MC_CONFIRM, by sending the contents of the local LU's send buffer and a confirmation request to the partner TP.

After this verb has successfully executed, the conversation identifier is no longer valid.

LU 6.2 Sync Point can use an optimization of the message flows known as implied forget. When the protocol specifies that a FORGET PS header is required, the next data flow on the session implies that a FORGET has been received. In the normal situation, the TP is aware of the next data flow when data is received or sent on one of its Sync Point conversations.

However, it is possible that the last message to flow is caused by the conversation being deallocated. In this case, the TP is unaware when the next data flow on the session occurs. In order to provide the TP with this notification, the **MC_DEALLOCATE** verb is modified to allow the TP to register a callback function which will be called:

- On the first normal flow transmission (request or response) over the session used by the conversation.
- If the session is unbound before any other data flows.
- If the session is terminated abnormally due to a DLC outage.

The **MC_DEALLOCATE** verb also contains a **correlator** field member that is returned as one of the parameters when the callback function is invoked. The application can use this parameter in any way (for example, as a pointer to a control block within the application).

The TP can use the *type* parameter passed to the callback function to determine whether the message flow indicates an implied forget has been received.

Note that the **MC_DEALLOCATE** verb will probably complete before the callback routine is called. The conversation is considered to be in RESET state and no further verbs can be issued using the conversation identifier. If the application issues a TP_ENDED verb before the next data flow on the session, the callback routine will not be invoked.

Host Integration Server 2009 allows TPs to deallocate conversations immediately after sending data by specifying the type parameter on MC_SEND_DATA as AP_SEND_DATA_DEALLOC_*. However, the **MC_SEND_DATA** verbs do not contain the implied forget callback function. TPs wishing to receive implied forget notification must issue **MC_DEALLOCATE** explicitly.

MC_FLUSH

The **MC_FLUSH** verb sends the contents of the local logical unit's (LU) send buffer to the partner LU and transaction program (TP). If the send buffer is empty, no action takes place.

The following structure describes the verb control block (VCB) used by the **MC_FLUSH** verb.

Syntax

```
struct mc_flush {
    unsigned short  opcode;
    unsigned char   opext;
    unsigned char   reserv2;
    unsigned short  primary_rc;
    unsigned long   secondary_rc;
    unsigned char   tp_id[8];
    unsigned long   conv_id;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_FLUSH.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_FLUSH_NOT_SEND_STATE

Secondary return code; the conversation was not in SEND state.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

Data processed by MC_SEND_DATA accumulates in the local LU's send buffer until one of the following happens:

- The local TP issues **MC_FLUSH** (or other verb that flushes the LU's send buffer).
- The buffer is full.

The request generated by MC_ALLOCATE is also buffered.

The conversation must be in SEND state when the TP issues this verb.

There is no state change.

MC_GET_ATTRIBUTES

The **MC_GET_ATTRIBUTES** verb returns the attributes of the conversation.

The following structure describes the verb control block (VCB) used by the **MC_GET_ATTRIBUTES** verb.

Syntax

```
struct mc_get_attributes {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    reserv3;
    unsigned char    sync_level;
    unsigned char    mode_name[8];
    unsigned char    net_name[8];
    unsigned char    lu_name[8];
    unsigned char    lu_alias[8];
    unsigned char    plu_alias[8];
    unsigned char    plu_un_name[8];
    unsigned char    reserv4[2];
    unsigned char    fqplu_name[17];
    unsigned char    reserv5;
    unsigned char    user_id[10];
    unsigned long    conv_group_id;
    unsigned char    conv_corr_len;
    unsigned char    conv_corr[8];
    unsigned char    reserv6[13];
//  NOTE: The following fields are present
//  when the high bit of opext is set
//  (opext & AP_EXTD_VCB) != 0.
    unsigned char    luw_id[26];
    unsigned char    sess_id[8];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_GET_ATTRIBUTES.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

sync_level

Returned parameter. Specifies the level of synchronization processing for the conversation. This parameter determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

AP_NONE indicates that confirmation processing will not be used in this conversation.

AP_CONFIRM_SYNC_LEVEL indicates that TPs can use confirmation processing in this conversation.

AP_SYNCPT indicates that TPs can use Sync Point Level 2 confirmation processing in this conversation.

mode_name

Returned parameter. Specifies the name of a set of networking characteristics. It is a type A EBCDIC character string.

net_name

Returned parameter. Specifies the name of the SNA network containing the local logical unit (LU) used by this TP. It is a type A EBCDIC character string.

lu_name

Returned parameter. Provides the name of the local LU.

lu_alias

Returned parameter. Provides the alias by which the local LU is known to the local TP. It is an ASCII character string.

plu_alias

Returned parameter. Provides the alias by which the partner LU is known to the local TP. It is an ASCII character string.

plu_un_name

Returned parameter. Specifies the uninterpreted name of the partner LU—the name of the partner LU as defined to the system services control point (SSCP). It is a type AE EBCDIC character string. This parameter is returned only if the local LU is dependent.

fqplu_name

Returned parameter. Provides the fully qualified name of the partner LU. It is a type A EBCDIC character string. The field contains the network name, an EBCDIC period, and the partner-LU name.

user_id

Returned parameter. Specifies the user identifier sent by the invoking TP through MC_ALLOCATE to access the invoked TP (if applicable). It is a type AE EBCDIC character string. The field contains the user identifier if the following conditions are true:

- The invoked TP requires conversation security.
- **MC_GET_ATTRIBUTES** was issued by the invoked TP.

Otherwise, the field contains spaces.

conv_group_id

Returned parameter. Specifies the conversation group identifier for the session to which the conversation has been allocated. This is also returned on MC_ALLOCATE and RECEIVE_ALLOCATE.

conv_corr_len

Returned parameter. Specifies the length of the conversation correlator identifier that is returned.

conv_corr

Returned parameter. Specifies the conversation correlator identifier (if any) that the source LU assigns to identify the conversation, which is unique for the source/partner LU pair. It is sent by the source LU on the allocation request.

Note

The following fields are present when the high bit of opext is set (opext & AP_EXTD_VCB) != 0.These fields are only present when using Sync Point Level 2 support.

luw_id

Logical unit-of-work identifier.

sess_id

Session identifier.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

MC_POST_ON_RECEIPT

The **MC_POST_ON_RECEIPT** verb allows the application to register to receive a notification when data or status arrives at the local logical unit (LU) without actually receiving it at the same time. This verb can only be issued while in RECEIVE state and it never causes a change in conversation state.

When the transaction program (TP) issues this verb, APPC returns control to the TP immediately. When the specified conditions are satisfied, the Win32® event specified by the **sema** parameter is signaled and the verb completes. Then the TP looks at the return code in the verb control block (VCB) to determine whether or not any data or status notification has arrived at the local LU and issues an MC_RECEIVE_IMMEDIATE or MC_RECEIVE_AND_WAIT verb to actually receive the data or status notification.

The **MC_POST_ON_RECEIPT** verb implements both the **POST_ON_RECEIPT** and **TEST** verbs as described in the IBM Transaction Programmer's manual for LU Type 6.2.

The following structure describes the verb control block used by the **MC_POST_ON_RECEIPT** verb.

Syntax

```
struct mc_post_on_receipt {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv1;
    unsigned char    primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned short   reserv2;
    unsigned char    reserv3;
    unsigned char    reserv4;
    unsigned short   max_len;
    unsigned short   reserv5;
    unsigned char *  reserv6;
    unsigned char    reserv7[5];
    unsigned long    sema;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_POST_ON_RECEIPT.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv1

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv2

A reserved field.

reserv3

A reserved field.

reserv4

A reserved field.

max_len

Supplied parameter. Specifies the length of data that triggers APPC to post a notification to the TP.

reserv5

A reserved field.

reserv6

A reserved field.

reserv7

A reserved field.

sema

Supplied parameter. Specifies the handle of a Win32 event. The event should have been created by the TP and the TP is responsible for ensuring that it is reset before a call is made and after the verb completes.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_DATA

Secondary return code; data is available for the program to receive.

AP_NOT_DATA

Secondary return code; information other than data is available for the program to receive.

AP_CANCELLED

Primary return code; the verb was canceled.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_INVALID_SEMAPHORE_HANDLE

Secondary return code; the **sema** parameter was not set to a valid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE.

- The partner TP has encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation and issued MC_DEALLOCATE with **dealloc_type** set to one of the following:

- AP_CONFIRM_SYNC_LEVEL
- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued MC_SEND_ERROR while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued MC_SEND_ERROR. Data sent but not yet received is purged.

AP_PROG_ERROR_TRUNC

Primary return code; the partner TP has issued MC_SEND_ERROR while the conversation was in SEND state. Data was truncated.

AP_SVC_ERROR_NO_TRUNC

Primary return code; the partner TP (or partner LU) issued MC_SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP was not truncated.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued MC_SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

AP_SVC_ERROR_TRUNC

Primary return code; the partner TP (or partner LU) issued MC_SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been truncated.

Remarks

While an **MC_POST_ON_RECEIPT** verb is outstanding, the following verbs can be issued on the same conversation:

GET_ATTRIBUTES

GET_TYPE

MC_DEALLOCATE

MC_RECEIVE_AND_WAIT

MC_RECEIVE_IMMEDIATE

MC_REQUEST_TO_SEND

MC_SEND_ERROR

MC_TEST_RTS

TP_ENDED

Issuing any of the following verbs prior to completion of the asynchronous **MC_POST_ON_RECEIPT** verb causes the

MC_POST_ON_RECEIPT verb to be canceled (the Win32 event is signaled and the primary return code in the verb control block is set to AP_CANCELLED).

MC_DEALLOCATE

MC_RECEIVE_AND_WAIT

MC_RECEIVE_IMMEDIATE

MC_SEND_ERROR

TP_ENDED

MC_PREPARE_TO_RECEIVE

The **MC_PREPARE_TO_RECEIVE** verb changes the state of the conversation for the local transaction program (TP) from SEND to RECEIVE.

The following structure describes the verb control block (VCB) used by the **MC_PREPARE_TO_RECEIVE** verb.

Syntax

```
struct mc_prepare_to_receive {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    primary_rc;
    unsigned short   reserv2;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    ptr_type;
    unsigned char    locks;
};
```

Remarks

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_PREPARE_TO_RECEIVE.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

ptr_type

Supplied parameter. Specifies how to perform the state change.

Use AP_FLUSH to send the contents of the local logical unit's (LU) send buffer to the partner LU (and TP) before changing the conversation's state to RECEIVE.

The AP_SYNC_LEVEL value uses the conversation's synchronization level (established by **MC_ALLOCATE**) to determine how to perform the state change.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before changing the conversation's state to RECEIVE. If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving

confirmation from the partner TP, APPC changes the conversation's state to RECEIVE. If, however, the partner TP reports an error, the state changes to RECEIVE or RESET. See the Remarks in this topic.

locks

Supplied parameter. Specifies when APPC should return control to the local TP.

Use this parameter only if **ptr_type** is set to AP_SYNC_LEVEL and the synchronization level of the conversation, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. (Otherwise, the parameter is ignored.)

- AP_LONG indicates that APPC returns control to the local TP when the confirmation and subsequent data from the partner TP arrive at the local LU. (This method results in more efficient use of the network but requires a longer time to return control to the local TP.)

- AP_SHORT indicates that APPC returns control to the local TP when the confirmation from the partner TP arrives at the local LU.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_P_TO_R_INVALID_TYPE

Secondary return code; the **ptr_type** parameter was not set to a valid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_P_TO_R_NOT_SEND_STATE

Secondary return code; the conversation was not in SEND state.

AP_P_TO_R_NOT_LL_BDY

Secondary return code; the local TP did not finish sending a logical record.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued MC_SEND_ERROR. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

Before changing the conversation state, this verb performs the equivalent of one of the following:

- MC_FLUSH, by sending the contents of the local LU's send buffer to the partner LU (and TP).
- MC_CONFIRM, by sending the contents of the local LU's send buffer and a confirmation request to the partner TP.

After this verb has successfully executed, the local TP can receive data.

The conversation must be in SEND state when the TP issues this verb.

State changes, summarized in the following table, are based on the value of **primary_rc**.

primary_rc	**New state**
AP_OK	RECEIVE
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

The conversation does not change to SEND state for the partner TP until the partner TP receives one of the following values through the **what_rcvd** parameter of a subsequent receive verb:

- AP_SEND
- AP_CONFIRM_SEND and replies with MC_CONFIRMED
- AP_DATA_COMPLETE_CONFIRM_SEND and replies with **MC_CONFIRMED**
- AP_DATA_CONFIRM_SEND and replies with **MC_CONFIRMED**

The receive verbs are MC_RECEIVE_AND_POST, MC_RECEIVE_IMMEDIATE, and MC_RECEIVE_AND_WAIT.

MC_RECEIVE_AND_POST

The **MC_RECEIVE_AND_POST** verb receives application data and status information asynchronously. This allows the local transaction program (TP) to proceed with processing while data is still arriving at the local logical unit (LU).

While an asynchronous **MC_RECEIVE_AND_POST** is outstanding, the following verbs can be issued on the same conversation:

- GET_TYPE
- MC_GET_ATTRIBUTES
- MC_REQUEST_TO_SEND
- MC_SEND_ERROR
- MC_TEST_RTS
- TP_ENDED

This allows an application to use an asynchronous **MC_RECEIVE_AND_POST** to receive data. While the **MC_RECEIVE_AND_POST** is outstanding, it can still use **MC_SEND_ERROR** and **REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support. For information on how a TP receives data and how to use this verb, see Remarks in this topic.

The following structure describes the verb control block (VCB) used by the **MC_RECEIVE_AND_POST** verb.

Syntax

```
struct mc_receive_and_post {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       reserv4;
    unsigned char       rts_rcvd;
    unsigned char       reserv5;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char FAR * sema;
    unsigned char       reserv6;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_RECEIVE_AND_POST.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary

depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Indicates whether data or conversation status was received.

- AP_CONFIRM_DEALLOCATE indicates that the partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL. The conversation's synchronization level, established by MC_ALLOCATE, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues MC_CONFIRMED.

- AP_CONFIRM_SEND indicates that the partner TP issued MC_PREPARE_TO_RECEIVE with ptr_type set to AP_SYNC_LEVEL. The conversation's synchronization level, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **MC_CONFIRMED**, and begins to send data.

- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP issued MC_CONFIRM. Upon receiving this value, the local TP normally issues **MC_CONFIRMED**.

- AP_DATA_COMPLETE indicates, for **MC_RECEIVE_AND_POST**, that the local TP has received a complete data record or the last part of a data record. Upon receiving this value, the local TP normally reissues **MC_RECEIVE_AND_POST** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data. Otherwise, the local TP examines status information.

  If **primary_rc** contains AP_OK and **what_rcvd** contains AP_SEND, AP_CONFIRM_SEND, AP_CONFIRM_DEALLOCATE, or AP_CONFIRM_WHAT_RECEIVED, see the description of the value (in this section) for the next action the local TP normally takes.

  If primary_rc contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to the MC_DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates, for **MC_RECEIVE_AND_POST**, that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record). Upon receiving this value, the local TP normally reissues **MC_RECEIVE_AND_POST** (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.

- AP_SEND indicates, for the partner TP, that the conversation has entered RECEIVE state. For the local TP, the conversation is now in SEND state. Upon receiving this value, the local TP normally uses MC_SEND_DATA to begin sending data.

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

- AP_NO specifies that indicators should be returned individually on separate invocations of the verb.

- AP_YES specifies that indicators should be returned together, provided both are available. Both can be returned when:

The receive buffer is large enough to hold all of the data that precedes the status indicator.

The data is the last data record before the status indicator.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued MC_REQUEST_TO_SEND.

- AP_YES indicates that the partner TP issued **MC_REQUEST_TO_SEND**, which requests that the local TP change the conversation to RECEIVE state.
- AP_NO indicates that the partner TP has not issued **MC_REQUEST_TO_SEND**.

Max_len

Supplied parameter. Specifies the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

The value must not exceed the length of the buffer to contain the received data. The offset of **dptr** plus the value of **max_len** must not exceed the size of the data segment.

dlen

Returned parameter. Specifies the number of bytes of data received. Data is stored in the buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Provides the address of the buffer to contain the data received by the local LU.

For the Microsoft® Windows® 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

For the OS/2 operating system, the data buffer must reside on an unnamed, shared segment, which is allocated by the **DosAllocSeg** function with Flags equal to 1. The data buffer must fit entirely on the data segment.

sema

Supplied parameter. Provides the address of the semaphore that APPC is to clear when the asynchronous receiving operation is finished. The **sema** parameter is an event handle obtained by calling either the **CreateEvent** or **OpenEvent** Win32 function.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_DEALLOC_NORMAL

Primary return code; the partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_FLUSH or AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE.

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_INVALID_SEMAPHORE_HANDLE

Secondary return code; the address of the RAM semaphore or system semaphore handle was invalid.

Note
APPC cannot trap all invalid semaphore handles. If the TP passes a bad RAM semaphore handle, a protection violation results.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_AND_POST_BAD_STATE

Secondary return code; the conversation was not in RECEIVE or SEND state when the TP issued this verb.

AP_RCV_AND_POST_NOT_LL_BDY

Secondary return code; the conversation was in SEND state; the TP began but did not finish sending a logical record.

AP_CANCELED

Primary return code; the local TP issued one of the following verbs, which canceled **MC_RECEIVE_AND_POST**:

MC_DEALLOCATE with **dealloc_type** set to AP_ABEND

MC_SEND_ERROR

TP_ENDED

Issuing one of these verbs causes the semaphore to be cleared.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the

allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP issued MC_SEND_ERROR while the conversation was in SEND state. Data was not

truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **MC_SEND_ERROR**. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received is one data record.

   The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it. The receive verbs are **MC_RECEIVE_AND_POST**, MC_RECEIVE_AND_WAIT, and MC_RECEIVE_IMMEDIATE.

2. The local TP issues the receive verb again. This has one of the following effects:

   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
   - If the partner TP has finished sending data or is waiting for confirmation, status information (available through **what_rcvd**) indicates the next action the local TP normally takes.

The following procedure shows tasks performed by the local TP in using **MC_RECEIVE_AND_POST**.

To use MC_RECEIVE_AND_POST

1. For the Windows® 2000 operating system, the TP retrieves the WinAsyncAPPC message number by calling the **RegisterWindowMessage** API or allocating a semaphore. The **sema** field should be set to NULL if the application expects to be notified through the Windows message mechanism.

   APPC sends the Windows message or clears the semaphore when the local TP finishes receiving data.

   For the OS/2 operating system, the TP uses the **DosSemSet** function to set the semaphore pointed to by **sema**.

   The semaphore will remain set while the local TP receives data asynchronously. APPC will clear the semaphore when the local TP finishes receiving data.

2. The TP issues **MC_RECEIVE_AND_POST**.
3. The TP checks the value of **primary_rc**.

If **primary_rc** is AP_OK, the receive buffer (pointed to by **dptr**) is asynchronously receiving data from the partner TP. While receiving data asynchronously, the local TP can:

- Perform tasks not related to this conversation.
- Issue MC_REQUEST_TO_SEND.
- Gather information about this conversation by issuing GET_TYPE, MC_GET_ATTRIBUTES, or MC_TEST_RTS.
- Prematurely cancel **MC_RECEIVE_AND_POST** by issuing MC_DEALLOCATE with **dealloc_type** set to AP_ABEND; MC_SEND_ERROR; or TP_ENDED.

If, however, **primary_rc** is not AP_OK, **MC_RECEIVE_AND_POST** has failed. In this case, the local TP does not perform the next two tasks.

4. For the Windows 2000 operating system, when the TP finishes receiving data asynchronously, APPC issues the WinAsyncAPPC Windows message or clears the semaphore.

   For the OS/2 operating system, the TP uses the **DosSemWait** function to wait for APPC to clear the semaphore pointed to by **sema**. When the TP finishes receiving data asynchronously, APPC clears the semaphore. To prevent the local TP from waiting, have it test the semaphore (invoking **DosSemWait** with **Timeout** set to zero) until APPC clears the semaphore.

5. The TP checks the new value of **primary_rc**.

   If **primary_rc** is AP_OK, the local TP can examine the other returned parameters and manipulate the asynchronously received data.

   If **primary_rc** is not AP_OK, only **secondary_rc** and **rts_rcvd** (request-to-send received) are meaningful.

Conversation State Effects

The conversation must be in RECEIVE or SEND state when the TP issues this verb.

Issuing **MC_RECEIVE_AND_POST** while the conversation is in SEND state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to PENDING_POST state; the local TP is ready to receive information from the partner TP asynchronously.

The conversation changes states twice:

- Upon initial return of the verb, if **primary_rc** contains AP_OK, the conversation changes to PENDING_POST state.
- After completion of the verb, the state changes depending on the value of the following:

  The **primary_rc** parameter

  The **what_rcvd** parameter if **primary_rc** is AP_OK

The following table shows the new state associated with each value of **what_rcvd** when **primary_rc** is AP_OK.

what_rcvd	New state
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE

AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	RECEIVE
AP_DATA_COMPLETE	RECEIVE
AP_DATA_INCOMPLETE	RECEIVE
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table shows the new state associated with each value of **primary_rc** other than AP_OK.

primary_rc	**New state**
AP_CANCELED	No change
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_PROG_ERROR_NO_TRUNC	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_NO_TRUNC	RECEIVE
AP_PROG_ERROR_TRUNC	RECEIVE
AP_SVC_ERROR_TRUNC	RECEIVE

MC_RECEIVE_AND_WAIT

The **MC_RECEIVE_AND_WAIT** verb receives any data that is currently available from the partner transaction program (TP). If no data is currently available, the local TP waits for data to arrive.

To allow full use to be made of the asynchronous support, asynchronously issued **MC_RECEIVE_AND_WAIT** verbs have been altered to act like MC_RECEIVE_AND_POST verbs. Specifically, while an asynchronous **MC_RECEIVE_AND_WAIT** is outstanding, the following verbs can be issued on the same conversation:

- GET_TYPE
- MC_GET_ATTRIBUTES
- MC_REQUEST_TO_SEND
- MC_SEND_ERROR
- MC_TEST_RTS
- TP_ENDED

This allows an application, and in particular, a 5250 emulator, to use an asynchronous **MC_RECEIVE_AND_WAIT** to receive data. While the **MC_RECEIVE_AND_WAIT** is outstanding, it can still use **MC_SEND_ERROR** and **MC_REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support.

The following structure describes the verb control block (VCB) used by the **MC_RECEIVE_AND_WAIT** verb.

Syntax

```
struct mc_receive_and_wait {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       reserv4;
    unsigned char       rts_rcvd;
    unsigned char       reserv5;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char       reserv6[5];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_RECEIVE_AND_WAIT.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Specifies the conversation identifier.

The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Indicates whether data or conversation status was received.

- AP_CONFIRM_DEALLOCATE indicates that the partner TP has issued MC_DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues MC_CONFIRMED.

- AP_CONFIRM_SEND indicates that the partner TP has issued MC_PREPARE_TO_RECEIVE with **ptr_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **MC_CONFIRMED** and begins to send data.

- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP has issued MC_CONFIRM. Upon receiving this value, the local TP normally issues **MC_CONFIRMED**.

- AP_DATA_COMPLETE indicates, for **MC_RECEIVE_AND_WAIT**, that the local TP has received a complete data record or the last part of a data record. Upon receiving this value, the local TP normally reissues **MC_RECEIVE_AND_WAIT** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data.

  Otherwise, the local TP examines status information, if **primary_rc** contains AP_OK and **what_rcvd** contains AP_SEND, AP_CONFIRM_SEND, AP_CONFIRM_DEALLOCATE, or AP_CONFIRM_WHAT_RECEIVED.

  See Return Codes in this topic for the next action the local TP normally takes.

  If primary_rc contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to MC_DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record). Upon receiving this value, the local TP normally reissues **MC_RECEIVE_AND_WAIT** (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.

- AP_SEND indicates, for the partner TP, that the conversation has entered RECEIVE state. For the local TP, the conversation is now in SEND state. Upon receiving this value, the local TP normally uses MC_SEND_DATA to begin sending data.

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

- AP_NO specifies that indicators should be returned individually on separate invocations of the verb.

- AP_YES specifies that indicators should be returned together, provided both are available. Both can be returned when:

  The receive buffer is large enough to hold all of the data that precedes the status indicator.

  The data is the last data record before the status indicator.

rts_rcvd

Returned parameter. Contains the request-to-send indicator.

- AP_YES indicates that the partner TP has issued MC_REQUEST_TO_SEND, which requests that the local TP change the conversation to RECEIVE state.

- AP_NO indicates that the partner TP has not issued **MC_REQUEST_TO_SEND**.

max_len

Supplied parameter. Indicates the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

For the Microsoft® Windows® 2000 operating system and the Windows graphical environment, this value must not exceed the length of the buffer to contain the received data.

For the OS/2 operating system, the offset of **dptr** plus the value of **max_len** must not exceed the size of the data segment.

By issuing **MC_RECEIVE_AND_WAIT** with **max_len** set to zero, the local TP can determine whether the partner TP has data to send, seeks confirmation, or has changed the conversation state.

dlen

Returned parameter. Indicates the number of bytes of data received. Data is stored in the buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Provides the address of the buffer to contain the data received by the local TP.

For the Windows 2000 operating system and the Windows graphical environment, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

For the OS/2 operating system, the data buffer must reside on an unnamed, shared segment, which is allocated by the **DosAllocSeg** function with Flags equal to 1. The data buffer must fit entirely on the data segment.

For the Windows environment, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Return Codes
AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation and issued MC_DEALLOCATE with **dealloc_type** set to one of the following:

- AP_CONFIRM_SYNC_LEVEL
- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_AND_WAIT_BAD_STATE

Secondary return code; the conversation was not in RECEIVE or SEND state when the TP issued this verb.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner logical unit (LU) or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued MC_SEND_ERROR while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **MC_SEND_ERROR**. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received is one data record.

   The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it.

   The receive verbs are MC_RECEIVE_AND_POST, **MC_RECEIVE_AND_WAIT**, and MC_RECEIVE_IMMEDIATE.

2. The local TP issues the receive verb again. This has one of the following effects:

   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
   - If the partner TP has finished sending data or is waiting for confirmation, status information (available through the **what_rcvd** parameter) indicates the next action the local TP normally takes.

The conversation must be in RECEIVE or SEND state when the TP issues this verb.

Issuing the Verb in SEND State

Issuing **MC_RECEIVE_AND_WAIT** while the conversation is in SEND state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to RECEIVE state; the local TP waits for the partner TP to send data.

State Change

The new conversation state is determined by the following factors:

- The state the conversation is in when the TP issues the verb.
- The **primary_rc** parameter.
- The **what_rcvd** parameter if **primary_rc** contains AP_OK.

Verb Issued in SEND State

The following table details the state changes when **MC_RECEIVE_AND_WAIT** is issued in SEND state and **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	RECEIVE
AP_DATA_COMPLETE	RECEIVE
AP_DATA_INCOMPLETE	RECEIVE
AP_SEND	No change
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when **MC_RECEIVE_AND_WAIT** is issued in SEND state and **primary_rc** is not AP_OK.

primary_rc	**New state**
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE

AP_PROG_ERROR_NO_TRUNC	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_NO_TRUNC	RECEIVE

Verb Issued in RECEIVE State

The following table details the state changes when **MC_RECEIVE_AND_WAIT** is issued in RECEIVE state and **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	No change
AP_DATA_COMPLETE	No change
AP_DATA_INCOMPLETE	No change
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when **MC_RECEIVE_AND_WAIT** is issued in RECEIVE state and **primary_rc** is not AP_OK.

primary_rc	**New state**
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET

AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	No change
AP_PROG_ERROR_NO_TRUNC	No change
AP_SVC_ERROR_PURGING	No change
AP_SVC_ERROR_NO_TRUNC	No change
AP_PROG_ERROR_TRUNC	No change
AP_SVC_ERROR_TRUNC	No change

MC_RECEIVE_IMMEDIATE

The **MC_RECEIVE_IMMEDIATE** verb receives any data currently available from the partner transaction program (TP). If no data is available, the local TP does not wait. To avoid blocking the conversation, issue MC_RECEIVE_AND_WAIT in conjunction with WinAsyncAPPC. The following structure describes the verb control block (VCB) used by the **MC_RECEIVE_IMMEDIATE** verb.

Syntax

```
struct mc_receive_immediate {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       reserv4;
    unsigned char       rts_rcvd;
    unsigned char       reserv5;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char       reserv6[5];
};
```

Members

Opcode

Supplied parameter. Specifies the verb operation code, AP_M_RECEIVE_IMMEDIATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

Reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Contains information received with the incoming data:

- AP_CONFIRM_DEALLOCATE indicates that the partner TP has issued MC_DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues MC_CONFIRMED.

- AP_CONFIRM_SEND indicates that the partner TP has issued MC_PREPARE_TO_RECEIVE with **ptr_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **MC_ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **MC_CONFIRMED**, and begins to send data.

- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP has issued MC_CONFIRM. Upon receiving this value, the local TP normally issues **MC_CONFIRMED**.

- AP_DATA_COMPLETE indicates, for **MC_RECEIVE_IMMEDIATE** in mapped conversations, that the local TP has received a complete data record or the last part of a data record. Upon receiving this value, the local TP normally reissues **MC_RECEIVE_IMMEDIATE** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data.

  Otherwise, the local TP examines status information if **primary_rc** contains AP_OK and **what_rcvd** contains any of these values:

  AP_SEND

  AP_CONFIRM_SEND

  AP_CONFIRM_DEALLOCATE

  AP_CONFIRM_WHAT_RECEIVED

  See the description of the value in Return Codes in this topic for the next action the local TP normally takes.

  If **primary_rc** contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to MC_DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates for **MC_RECEIVE_IMMEDIATE** in mapped conversations that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record). Upon receiving this value, the local TP normally reissues **MC_RECEIVE_IMMEDIATE** (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.

- AP_SEND indicates, for the partner TP, the conversation has entered RECEIVE state. For the local TP, the conversation is now in SEND state. Upon receiving this value, the local TP normally uses MC_SEND_DATA to begin sending data.

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

- AP_NO specifies that indicators should be returned individually on separate invocations of the verb.

- AP_YES specifies that indicators should be returned together, provided both are available. Both can be returned when:

  The receive buffer is large enough to hold all of the data that precedes the status indicator.

  The data is the last data record before the status indicator.

rts_rcvd

Returned parameter. Contains the request-to-send indicator. Possible values are:

- AP_YES indicates that the partner TP has issued MC_REQUEST_TO_SEND, which requests that the local TP change the

conversation to RECEIVE state.

- AP_NO indicates that the partner TP has not issued **MC_REQUEST_TO_SEND**.

max_len

Supplied parameter. Indicates the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

For the Microsoft® Windows® 2000 operating system and the Windows graphical environment, this value must not exceed the length of the buffer to contain the received data.

For the OS/2 operating system, the offset of **dptr** plus the value of **max_len** must not exceed the size of the data segment.

By issuing **MC_RECEIVE_IMMEDIATE** with **max_len** set to zero, the local TP can determine whether the partner TP has data to send, seeks confirmation, or has changed the conversation state.

dlen

Returned parameter. Provides the number of bytes of data received. Data is stored in a buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Address of the buffer to contain the data received by the local TP.

For the Windows 2000 operating system and the Windows graphical environment, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

For the OS/2 operating system, the data buffer must reside on an unnamed, shared segment, which is allocated by the function **DosAllocSeg** with Flags equal to 1. The data buffer must fit entirely on the data segment.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_UNSUCCESSFUL

Primary return code; no data is immediately available from the partner TP.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation. The partner TP issued MC_DEALLOCATE with **dealloc_type** set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_IMMD_BAD_STATE

Secondary return code; the conversation was not in RECEIVE state.

AP_ALLOCATION_ERROR

Secondary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner logical unit (LU) or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued MC_SEND_ERROR while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **MC_SEND_ERROR**. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received is one data record.

   The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it.

   The receive verbs are MC_RECEIVE_AND_POST, MC_RECEIVE_AND_WAIT, and **MC_RECEIVE_IMMEDIATE**.

2. The local TP issues the receive verb again. This has one of the following effects:

   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
   - If the partner TP has finished sending data or is waiting for confirmation, status information (available through **what_rcvd**) indicates the next action the local TP normally takes.

The conversation must be in RECEIVE state when the TP issues this verb.

The new state is determined by **primary_rc**. If **primary_rc** is AP_OK, the new state is determined by **what_rcvd**.

The following table details the state changes when the **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	No change
AP_DATA_COMPLETE	No change
AP_DATA_INCOMPLETE	No change
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when the **primary_rc** is not AP_OK.

primary_rc	New state
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	No change
AP_PROG_ERROR_NO_TRUNC	No change
AP_SVC_ERROR_PURGING	No change
AP_SVC_ERROR_NO_TRUNC	No change
AP_PROG_ERROR_TRUNC	No change
AP_SVC_ERROR_TRUNC	No change
AP_UNSUCCESSFUL	No change

MC_RECEIVE_LOG_DATA

The **MC_RECEIVE_LOG_DATA** verb allows the user to register to receive the log data associated with an inbound Function Management Header 7 (FMH7) error report. The verb passes a buffer to APPC, and any log data received is placed in that buffer. APPC continues to use this buffer as successive FMH7s arrive until it is provided with another buffer (that is, until the transaction program (TP) issues another **MC_RECEIVE_LOG_DATA** specifying a different buffer or no buffer at all).

Note that the TP itself is responsible for allocating and freeing the buffer. After the buffer has been passed to APPC, the TP should either issue another **MC_RECEIVE_LOG_DATA** specifying a new buffer or a zero-length buffer, or wait until the conversation has finished before freeing the original buffer.

When an FMH7 is received, APPC copies any associated error log general data stream (GDS) into the buffer. If there is no associated error log variable, the buffer is zeroed out. It is up to the TP to check the buffer whenever a return code from a receive verb indicates that an error has been received.

The following structure describes the verb control block (VCB) used by the **MC_RECEIVE_LOG_DATA** verb.

Syntax

```
struct mc_receive_log_data {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv1;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      log_dlen;
    unsigned char FAR * log_dptr;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_RECEIVE_LOG_DATA.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv1

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

log_dlen

Supplied parameter. Specifies the maximum length of log data that APPC can place in the buffer (that is, the buffer size). The range is from 0 through 65535. Note that a length of zero here indicates that any previous **MC_RECEIVE_LOG_DATA** verb

should be cancelled.

log_dptr

Supplied parameter. Specifies the address of the buffer that APPC will use to store the log data.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

MC_REQUEST_TO_SEND

The **MC_REQUEST_TO_SEND** verb notifies the partner transaction program (TP) that the local TP wants to send data.

The following structure describes the verb control block (VCB) used by the **MC_REQUEST_TO_SEND** verb.

Syntax

```
struct mc_request_to_send {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_REQUEST_TO_SEND.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_R_T_S_BAD_STATE

Secondary return code; the conversation is not in an allowed state when the TP issued this verb.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Microsoft Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any of the following states when the TP issues this verb:

CONFIRM

PENDING_POST (OS/2)

RECEIVE

There is no state change.

The request-to-send notification is received by the partner program through the **rts_rcvd** parameter of the following verbs:

- MC_CONFIRM
- MC_RECEIVE_AND_POST
- MC_RECEIVE_AND_WAIT
- MC_RECEIVE_IMMEDIATE
- MC_SEND_DATA
- MC_SEND_ERROR

It is also indicated by a **primary_rc** of AP_OK on MC_TEST_RTS.

Request-to-send notification is sent to the partner TP immediately; APPC does not wait until the send buffer fills up or is flushed. Consequently, the request-to-send notification may arrive out of sequence. For example, if the local TP is in SEND state and issues MC_PREPARE_TO_RECEIVE followed by **MC_REQUEST_TO_SEND**, the partner TP, in RECEIVE state, may receive the request-to-send notification before it receives the send notification. For this reason, request-to-send can be reported to a TP through a receive verb.

In response to this request, the partner TP can change the conversation to:

- RECEIVE state by issuing **MC_PREPARE_TO_RECEIVE** or MC_RECEIVE_AND_WAIT.
- PENDING_POST state by issuing MC_RECEIVE_AND_POST.

The partner TP can also ignore the request-to-send.

The conversation state changes to SEND for the local TP when the local TP receives one of the following values through the **what_rcvd** parameter of a subsequent receive verb:

- AP_CONFIRM_SEND and replies with MC_CONFIRMED
- AP_DATA_COMPLETE_CONFIRM_SEND and replies with **MC_CONFIRMED**
- AP_SEND

The receive verbs are MC_RECEIVE_AND_POST, MC_RECEIVE_IMMEDIATE, and MC_RECEIVE_AND_WAIT.

MC_SEND_CONVERSATION

The **MC_SEND_CONVERSATION** verb allocates a session between the local logical unit (LU) and partner LU, sends data on the session, and then deallocates the session.

The following structure describes the verb control block (VCB) used by the **MC_SEND_CONVERSATION** verb.

Syntax

```
struct mc_send_conversation {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       reserv3[8];
    unsigned char       rtn_ctl;
    unsigned char       reserv4;
    unsigned long       conv_group_id;
    unsigned long       sense_data;
    unsigned char       plu_alias[8];
    unsigned char       mode_name[8];
    unsigned char       tp_name[64];
    unsigned char       security;
    unsigned char       reserv6[11];
    unsigned char       pwd[10];
    unsigned char       user_id[10];
    unsigned short      pip_dlen;
    unsigned char FAR * pip_dptr;
    unsigned char       reserv6;
    unsigned char       fqplu_name[17];
    unsigned char       reserv7[8];
    unsigned short      dlen;
    unsigned char FAR * dptr;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_SEND_CONVERSATION.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter was returned by TP_STARTED.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

rtn_ctl

Supplied parameter. Specifies how APPC should select a session to allocate for the conversation and when the local LU should return control to the local TP. The allowed values are:

- AP_IMMEDIATE specifies that the LU allocates a contention-winner session, if one is immediately available, and returns control to the TP.
- AP_WHEN_SESSION_ALLOCATED specifies that the LU does not return control to the TP until it allocates a session or encounters one of the errors described in Return Codes in this topic. If the session limit is zero, the LU returns control immediately. Note that if a session is not available, the TP waits for one.
- AP_WHEN_SESSION_FREE specifies that the LU allocates a contention-winner or contention-loser session, if one is available or able to be activated, and returns control to the TP. If an error occurs (as described in Return Codes in this topic) the call will return immediately with the error in the **primary_rc** and **secondary_rc** fields.
- AP_WHEN_CONWINNER_ALLOC specifies that the LU does not return control until it allocates a contention-winner session or encounters one of the errors described in Return Codes in this topic. If the session limit is zero, the LU returns control immediately. Note that if a session is not available, the TP waits for one.
- AP_WHEN_CONV_GROUP_ALLOC specifies that the LU does not return control to the TP until it allocates the session specified by **conv_group_id** or encounters one of the errors described in Return Codes in this topic. If the session is not available, the TP waits for it to become free.

conv_group_id

Supplied/returned parameter. Used as a supplied parameter when **rtn_ctl** is WHEN_CONV_GROUP_ALLOC to specify the identity of the conversation group from which the session should be allocated. When **rtn_ctl** specifies a different value, and the **primary_rc** is AP_OK, this is a returned value. The purpose of this parameter is to provide a TP with the assurance that the same session will be reallocated and therefore the conversations conducted over the session will occur in the same sequence that they were initiated.

sense_data

Returned parameter. If the primary and secondary return codes indicate an allocation error (retry or no-retry), an SNA-defined sense code is returned.

plu_alias

Supplied parameter. Specifies the alias by which the partner LU is known to the local TP. This parameter must match the name of a partner LU established during configuration. The parameter is an 8-byte, type G ASCII character set that includes:

- Uppercase letters
- Numerals 0 to 9
- Spaces
- Special characters $, #, %, and @

If the value of this parameter is fewer than eight bytes, pad it on the right with ASCII spaces (0x20).

mode_name

Supplied parameter. Specifies the name of a set of networking characteristics defined during configuration. This parameter must match the name of a mode associated with the partner LU during configuration.

The parameter is an 8-byte EBCDIC character string. It can consist of characters from the type A EBCDIC character set, including all EBCDIC spaces. These characters are:

- Uppercase letters

- Numerals 0 to 9
- Special characters $, #, and @

The first character in the string must be an uppercase letter or special character.

In a mapped conversation, the name cannot be SNASVCMG (a reserved mode name used internally by APPC).

tp_name

Supplied parameter. Specifies the name of the invoked TP. The value of **tp_name** specified by MC_ALLOCATE in the invoking TP must match the value of **tp_name** specified by RECEIVE_ALLOCATE in the invoked TP.

The parameter is a 64-byte, case-sensitive, EBCDIC character string. This parameter can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention is that a service TP name can have up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the EBCDIC AE character set.

security

Supplied parameter. Specifies the information the partner LU requires in order to validate access to the invoked TP.

- AP_NONE specifies that the invoked TP uses no conversation security.
- AP_PGM specifies that the invoked TP uses conversation security and requires a user identifier and password. Use **user_id** and **pwd** to supply this information.
- AP_SAME specifies that the invoked TP, invoked with a valid user identifier and password, in turn invokes another TP.

For example, assume that TP A invokes TP B with a valid user identifier and password, and TP B in turn invokes TP C. If TP B specifies the value AP_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator indicates to TP C not to require the password (if TP C is configured to accept an already-verified indicator).

pwd

Supplied parameter. Specifies the password associated with **user_id**. This parameter is required only if the security parameter is set to AP_PGM and must match the password for **user_id** that was established during configuration.

This parameter is a 10-byte, case-sensitive, EBCDIC character string. It can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

user_id

Supplied parameter. Specifies the user identifier required to access the partner TP. This parameter is required only if the security parameter is set to AP_PGM and must match one of the user identifiers configured for the partner TP.

The parameter can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the user identifier is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

pip_dlen

Supplied parameter. Specifies the length of the PIP to be passed to the partner TP. The range for this parameter is from 0 through 32767.

pip_dptr

Supplied parameter. Specifies the address of the buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

PIP data can consist of initialization parameters or environmental setup information required by a partner TP or remote operating system. The PIP data must follow the GDS format. For more information, see your IBM SNA manual(s).

For the Microsoft® Windows® 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area.

fqplu_name

Supplied parameter. Specifies the fully qualified name of the local LU. This parameter must match the fully qualified name of the local LU defined in the remote node. The parameter is made up of two type A EBCDIC character strings (each of up to eight characters), which are the network name (NETID) and the LU name of the partner LU. The names are separated by an EBCDIC period (.). The NETID can be omitted, and if this is the case, the period should also be omitted.

This name must be provided if no **plu_alias** is provided.

Type A EBCDIC characters contain:

- Uppercase letters
- Numerals 0 to 9
- Special characters $, #, and @

If the value of this parameter is fewer than 17 bytes, pad it on the right with EBCDIC spaces (0x40).

dlen

Supplied parameter. Specifies the number of bytes of data to be put in the local LU's send buffer. The range for this parameter is from 0 through 65535.

dptr

Supplied parameter. Specifies the address of the buffer containing the data to be put in the local LU's send buffer.

For the Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; the supplied parameter **rtn_ctl** specified immediate return of the control to the TP (AP_IMMEDIATE), and the local LU did not have an available contention-winner session.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_RETURN_CONTROL

Secondary return code; the value specified for **rtn_ctl** was invalid.

AP_BAD_SECURITY

Secondary return code; the value specified for **security** was invalid.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_PIP_LEN_INCORRECT

Secondary return code; the value of **pip_dlen** was greater than 32767.

AP_UNKNOWN_PARTNER_MODE

Secondary return code; the value specified for **mode_name** was invalid.

AP_BAD_PARTNER_LU_ALIAS

Secondary return code, APPC did not recognize the supplied **partner_lu_alias**.

AP_NO_USE_OF_SNASVCMG

Secondary return code; SNASVCMG is not a valid value for **mode_name**.

AP_INVALID_DATA_SEGMENT

Secondary return code; the PIP data or application data was longer than the allocated data segment, or the address of a data buffer was wrong.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can

satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

This verb is issued by the invoking TP to conduct an entire conversation with the remote TP. If the remote TP rejects either the conversation initiation or the data, the invoking TP will not receive notification of the rejection.

The conversation state is RESET when the TP issues this verb. There is no state change.

Several parameters of **MC_SEND_CONVERSATION** are EBCDIC or ASCII strings. A TP can use the common service verb (CSV) CONVERT to translate a string from one character set to the other.

Normally, the value of **mode_name** must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU. If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**.

MC_SEND_DATA

The **MC_SEND_DATA** verb places data in the local logical unit's (LU) send buffer for transmission to the partner transaction program (TP).

The following structure describes the verb control block (VCB) used by the **MC_SEND_DATA** verb.

Syntax

```
struct mc_send_data {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       rts_rcvd;
    unsigned char       data_type;
    unsigned short int  dlen;
    unsigned char FAR * dptr ;
    unsigned char       type;
    unsigned char       reserv4;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_SEND_DATA.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Provides the request-to-send-received indicator.

- AP_YES indicates that the partner TP has issued MC_REQUEST_TO_SEND, which requests that the local TP change the conversation to RECEIVE state. To change to RECEIVE state, the local TP can use MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, or MC_RECEIVE_AND_POST.

- AP_NO indicates that the partner TP has not issued **MC_REQUEST_TO_SEND**.

data_type

Supplied parameter. Specifies the type of data to be sent if Sync Point is supported. Valid parameters are:

AP_APPLICATION

AP_USER_CONTROL_DATA

AP_PS_HEADER

dlen

Supplied parameter. Specifies the number of bytes of data to be put in the local LU's send buffer. The range is from 0 through 65535.

dptr

Supplied parameter. Specifies the address of the buffer containing the data to be put in the local LU's send buffer.

For the Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

type

Supplied parameter. Allows a TP to send data and perform other functions within one API call. For example, you can combine **MC_SEND_DATA** with **type** set to CONFIRM to accomplish the same objective as issuing **MC_SEND_DATA** followed by MC_CONFIRM.

- AP_SEND_DATA_CONFIRM corresponds to **MC_SEND_DATA** followed by **MC_CONFIRM**.
- AP_SEND_DATA_FLUSH corresponds to **MC_SEND_DATA** followed by MC_FLUSH.
- AP_SEND_DATA_DEALLOC_ABEND corresponds to **MC_SEND_DATA** followed by MC_DEALLOCATE with a **dealloc_type** of AP_ABEND.
- AP_SEND_DATA_DEALLOC_FLUSH corresponds to **MC_SEND_DATA** followed by **MC_DEALLOCATE** with a **dealloc_type** of AP_FLUSH.
- AP_SEND_DATA_DEALLOC_SYNC_LEVEL corresponds to **MC_SEND_DATA** followed by **MC_DEALLOCATE** with a **dealloc_type** of AP_SYNC_LEVEL.
- AP_SEND_DATA_P_TO_R_FLUSH corresponds to **MC_SEND_DATA** followed by MC_PREPARE_TO_RECEIVE with a **ptr_type** of AP_FLUSH.
- AP_SEND_DATA_P_TO_R_SYNC_LEVEL corresponds to **MC_SEND_DATA** followed by **MC_PREPARE_TO_RECEIVE** with a **ptr_type** of AP_SYNC_LEVEL and **locks** set to AP_SHORT.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_SEND_DATA_INVALID_TYPE

Secondary return code; the specified type was not recognized by APPC.

AP_SEND_DATA_CONFIRM_SYNC_NONE

Secondary return code; the **type** CONFIRM is not permitted for a conversation that was allocated with a **sync_level** of NONE.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_SEND_DATA_NOT_SEND_STATE

Secondary return code; the local TP issued **MC_SEND_DATA**, but the conversation was not in SEND state.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Microsoft Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **MC_SEND_ERROR**. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

Remarks

The conversation must be in SEND state when the TP issues this verb. State changes, based on **primary_rc**, are summarized in the following table.

primary_rc	**New state**
AP_OK	No change
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

MC_SEND_DATA may wait indefinitely because the partner TP has not issued a receive verb. If this occurs, the send buffer may fill up.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) when one of the following occurs:

- The send buffer fills up.
- The local TP issues MC_FLUSH, MC_CONFIRM, or MC_DEALLOCATE (or other verb that flushes the LU's send buffer).

MC_SEND_ERROR

The **MC_SEND_ERROR** verb notifies the partner transaction program (TP) that the local TP has encountered an application-level error.

The following structure describes the verb control block (VCB) used by the **MC_SEND_ERROR** verb.

Syntax

```
struct mc_send_error {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       rts_rcvd;
      unsigned char       err_type;
    unsigned char       err_dir;
    unsigned char       reserv4;
    unsigned char       reserv5[2];
    unsigned char       reserv6[4];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_SEND_ERROR.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by MC_ALLOCATEin the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued MC_REQUEST_TO_SEND.Possible values include:

- AP_YES indicates that the partner TP has issued **MC_REQUEST_TO_SEND**, which requests that the local TP change the conversation to RECEIVE state. To change to RECEIVE state, the local TP can use MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, or MC_RECEIVE_AND_POST.

- AP_NO indicates that the partner TP has not issued **MC_REQUEST_TO_SEND**.

err_type

For a mapped conversation, this parameter is supplied if Sync Point is supported. Valid values are:

AP_PROG

AP_BACKOUT_NO_RESYNC

AP_BACKOUT_RESYNC

err_dir

Supplied parameter. Indicates whether the error is with data just received or with data that is about to be sent. Use this parameter only when the conversation is in SEND_PENDING state. The parameter is ignored otherwise. The following are allowed values:

- AP_RCV_DIR_ERROR indicates that the TP issued **MC_SEND_ERROR** after detecting an error associated with the data just received.
- AP_SEND_DIR_ERROR indicates that the TP issued **MC_SEND_ERROR** after detecting an error associated with data it was going to send. For example, the TP encountered an error while reading data from the disk drive.

reserv3

A reserved field.

Return Codes
AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_ERROR_DIRECTION

Secondary return code; the specified **err_dir** was not recognized by APPC.

AP_SEND_ERROR_BAD_TYPE

Secondary return code; the value of **err_type** was invalid.

AP_SEND_ERROR_LOG_LL_WRONG

Secondary return code; the LL field of the error log GDS variable did not match the actual length of the data.

The following return codes can be generated when **MC_SEND_ERROR** is issued in any allowed state:

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Microsoft Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This may occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

The following return codes can be generated only if **MC_SEND_ERROR** is issued in SEND state:

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after MC_ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **MC_SEND_ERROR**. Data sent but not yet received is purged.

AP_DEALLOC_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner LU to send an **MC_DEALLOCATE** request.

The following return code can be generated only if **MC_SEND_ERROR** is issued in RECEIVE state:

AP_DEALLOC_NORMAL

Primary return code; this return code does not indicate an error.

The partner TP issued MC_DEALLOCATE with **dealloc_type** set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

Remarks

The conversation can be in any state except RESET when the TP issues this verb. The conversation state must be SEND_PENDING if **err_dir** is used.

The local TP sends the error notification immediately to the partner TP; it does not hold the information in the local LU's send buffer.

Upon successful execution of this verb, the conversation is in SEND state for the local TP and in RECEIVE state for the partner TP.

The new state is determined by **primary_rc**. Possible state changes are summarized in the following table.

primary_rc	**New state**
AP_OK	SEND
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

If the conversation is in RECEIVE state when the TP issues **MC_SEND_ERROR**, incoming data is purged by APPC. This data includes:

- Data sent by MC_SEND_DATA.
- Return code indicators.
- Confirmation requests.
- Deallocation requests.

APPC does not purge an incoming request-to-send indicator. APPC replaces purged incoming return code indicators with other return codes. The primary return code AP_OK replaces the following purged return code indicators:

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING

AP_PROG_ERROR_TRUNC

AP_SVC_ERROR_NO_TRUNC

AP_SVC_ERROR_PURGING

AP_SVC_ERROR_TRUNC

The primary return code AP_DEALLOC_NORMAL replaces the following purged return code indicators:

AP_ALLOCATION_ERROR

AP_ALLOCATION_FAILURE_NO_RETRY

AP_ALLOCATION_FAILURE_RETRY

AP_CONVERSATION_TYPE_MISMATCH

AP_DEALLOC_ABEND

AP_DEALLOC_ABEND_PROG

AP_DEALLOC_ABEND_SVC

AP_DEALLOC_ABEND_TIMER

AP_PIP_NOT_ALLOWED

AP_PIP_NOT_SPECIFIED_CORRECTLY

AP_SECURITY_NOT_VALID

AP_SYNC_LEVEL_NOT_SUPPORTED

AP_TP_NAME_NOT_RECOGNIZED

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

AP_TRANS_PGM_NOT_AVAIL_RETRY

When the conversation is in SEND_PENDING state, APPC reports the following return codes to the partner TP based on the value in **err_dir**:

AP_PROG_ERROR_PURGING

The local TP issued **MC_SEND_ERROR** with RECEIVE as the **err_dir**.

AP_PROG_ERROR_NO_TRUNC

The local TP issued **MC_SEND_ERROR** with SEND as the **err_dir**.

MC_TEST_RTS

The **MC_TEST_RTS** verb determines whether a request-to-send notification has been received from the partner transaction program (TP).

The following structure describes the verb control block (VCB) used by the **MC_TEST_RTS** verb.

Syntax

```
struct mc_test_rts {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       reserv3;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_TEST_RTS.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter was returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; request-to-send notification has not been received.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

MC_TEST_RTS_AND_POST

The **MC_TEST_RTS_AND_POST** verb allows an application, typically a 5250 emulator, to request asynchronous notification when a partner transaction program (TP) requests send direction.

The following structure describes the verb control block (VCB) used by the **MC_TEST_RTS_AND_POST** verb.

Syntax

```
struct mc_test_rts_and_post {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       reserv3;
    unsigned long       handle;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_M_TEST_RTS_AND_POST.

opext

Supplied parameter. Specifies the verb operation extension, AP_MAPPED_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter was returned by MC_ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

handle

Supplied parameter. On Microsoft® Windows® 2000, this field provides the event handle to set.

Return Codes from Initial Verb

AP_OK

Primary return code; the verb executed successfully. Note particularly that a return code of AP_OK from the initial verb does not indicate that **MC_REQUEST_TO_SEND** verb received from the partner TP. It simply indicates that the facility to receive asynchronous notification has been registered.

AP_UNSUCCESSFUL

Primary return code; request-to-send notification has not been received.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with MC_ALLOCATE, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **MC_ALLOCATE** request.

When **MC_ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Return Codes from Asynchronous Completion
AP_OK

Primary return code; the request-to-send notification has been received from the partner TP.

AP_CANCELLED

The outstanding **TEST_RTS_AND_POST** verb has been terminated. This will occur if the underlying conversation has been deallocated or an AP_TP_ENDED has been issued. Note that as with **RECEIVE_AND_POST**, the TP is still responsible for correctly terminating the conversation and possibly terminating the TP. Issuing another verb, such as **RECEIVE_IMMEDIATE,** at this point will indicate the reason for the conversation failure.

Remarks

The conversation can be in any state except RESET when the TP issues this verb. There is no state change.

A common feature of many APPC applications, such as 5250 emulators, is a requirement to detect a partner's request to send. Currently, this can be done by polling the APPC interface to detect the partner's request. For example, an application can occasionally issue one of the following verbs:

- **MC_TEST_RTS**
- **MC_RECEIVE_IMMEDIATE** and check the **rts_rcvd** field
- **MC_SEND_DATA** of zero bytes, again checking the **rts_rcvd** field.

Some of the problems associated with this polling approach are:

- The application must continually interrupt its main work to poll APPC.
- The partner's request is not detected as soon as it becomes available.
- These approaches are processor-intensive.

The **MC_TEST_RTS_AND_POST** verb allows an application running on Windows 2000, typically a 5250 emulator, to request asynchronous notification when the partner TP requests send direction.

An APPC application typically issues the **MC_TEST_RTS_AND_POST** verb while in SEND state and then continues with its main processing. A request for send direction from the partner TP is indicated asynchronously to the application. After dealing with the partner's request, the application typically returns to SEND state, reissues **MC_TEST_RTS_AND_POST**, and continues.

The **MC_TEST_RTS_AND_POST** verb completes synchronously and the return code AP_OK indicates that a request for asynchronous notification has been registered. It is important to emphasize that this does not indicate that request-to-send was received from the partner TP.

When the partner's request to send is received, the asynchronous event completion occurs. It is important to note that this may be before the completion of the local TP's original **MC_TEST_RTS_AND_POST** verb. This will be the case if the partner's request to send was received before the local TP's **MC_TEST_RTS_AND_POST** verb was issued, or while the local TP's **MC_TEST_RTS_AND_POST** verb was being processed.

POST_ON_RECEIPT

The **POST_ON_RECEIPT** verb allows the application to register to receive a notification when data or status arrives at the local logical unit (LU) without actually receiving it at the same time. This verb can only be issued while in RECEIVE state and it never causes a change in conversation state. This verb is only supported on Microsoft® Windows® 2000.

When the transaction program (TP) issues this verb, APPC returns control to the TP immediately. When the specified conditions are satisfied the Win32® event specified by the **sema** parameter is signaled and the verb completes. Then the TP looks at the return code in the verb control block to determine whether or not any data or status notification has arrived at the local LU and issues a RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT verb to actually receive the data or status notification.

The **POST_ON_RECEIPT** verb implements both the **POST_ON_RECEIPT** and **TEST** verbs as described in the IBM Transaction Programmer's manual for LU Type 6.2.

The following structure describes the verb control block (VCB) used by the **POST_ON_RECEIPT** verb.

Syntax

```
struct post_on_receipt {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv1;
    unsigned char    primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned short   reserv2;
    unsigned char    fill;
    unsigned char    reserv4;
    unsigned short   max_len;
    unsigned short   reserv5;
    unsigned char *  reserv6;
    unsigned char    reserv7[5];
    unsigned long    sema;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_POST_ON_RECEIPT.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv1

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATEin the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv2

A reserved field.

fill

Supplied parameter. Specifies how the local TP receives data. The following values are allowed:

AP_BUFFER

Specifies that APPC should post a notification when the number of data bytes specified by **max_len** have arrived at the local LU, the end of the data has been reached, or information other than data is received (such as a conversation status, a confirmation, or a syncpoint request).

AP_LL

Specifies that APPC should post a notification when a complete or truncated logical record is received, when a portion of a logical record is received which is at least equal in length to the length specified by **max_len**, or when information other than data is received.

reserv4

A reserved field.

max_len

Supplied parameter. Specifies the length of data that triggers APPC to post a notification to the TP.

reserv5

A reserved field.

reserv6

A reserved field.

reserv7

A reserved field.

sema

Supplied parameter. Specifies the handle of a Win32 event. The event should have been created by the TP and the TP is responsible for ensuring that it is reset before a call is made and after the verb completes.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_DATA

Secondary return code; data is available for the program to receive.

AP_NOT_DATA

Secondary return code; information other than data is available for the program to receive.

AP_CANCELLED

Primary return code; the verb was canceled.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_INVALID_SEMAPHORE_HANDLE

Secondary return code; the **sema** parameter was not set to a valid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued DEALLOCATE with **dealloc_type**set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation and issued DEALLOCATE with **dealloc_type** set to one of the following:

- AP_CONFIRM_SYNC_LEVEL
- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued SEND_ERROR while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued SEND_ERROR. Data sent but not yet received is purged.

AP_PROG_ERROR_TRUNC

Primary return code; the partner TP has issued SEND_ERROR while the conversation was in SEND state. Data was truncated.

AP_SVC_ERROR_NO_TRUNC

Primary return code; the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP was not truncated.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

AP_SVC_ERROR_TRUNC

Primary return code; the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been truncated.

Remarks

While a **POST_ON_RECEIPT** verb is outstanding, the following verbs can be issued on the same conversation:

GET_ATTRIBUTES

GET_TYPE

DEALLOCATE

RECEIVE_AND_WAIT

RECEIVE_IMMEDIATE

REQUEST_TO_SEND

SEND_ERROR

TEST_RTS

TP_ENDED

Issuing any of the following verbs prior to completion of the asynchronous **POST_ON_RECEIPT** verb causes the **POST_ON_RECEIPT** verb to be canceled (the Win32 event is signaled and the primary return code in the verb control block is set to AP_CANCELLED).

DEALLOCATE

RECEIVE_AND_WAIT

RECEIVE_IMMEDIATE

SEND_ERROR

TP_ENDED

PREPARE_TO_RECEIVE

The **PREPARE_TO_RECEIVE** verb changes the state of the conversation for the local transaction program (TP) from SEND to RECEIVE.

The following structure describes the verb control block (VCB) used by the **PREPARE_TO_RECEIVE** verb.

Syntax

```
struct prepare_to_receive {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    primary_rc;
    unsigned short   reserv2;
    unsigned long    secondary_rc;
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    ptr_type;
    unsigned char    locks;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_PREPARE_TO_RECEIVE.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATEin the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

ptr_type

Supplied parameter. Specifies how to perform the state change.

Use AP_FLUSH to send the contents of the local logical unit's (LU) send buffer to the partner LU (and TP) before changing the conversation's state to RECEIVE.

The AP_SYNC_LEVEL value uses the conversation's synchronization level (established by **ALLOCATE**) to determine how to perform the state change.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before changing the conversation's state to RECEIVE. If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC changes the conversation's state to RECEIVE. If, however, the partner TP reports an error, the state changes to RECEIVE or RESET. See the Remarks in this topic.

locks

Supplied parameter. Specifies when APPC should return control to the local TP.

Use this parameter only if **ptr_type** is set to AP_SYNC_LEVEL and the synchronization level of the conversation, established by ALLOCATE, is AP_CONFIRM_SYNC_LEVEL. (Otherwise, the parameter is ignored.)

Use AP_LONG to indicate that APPC returns control to the local TP when the confirmation and subsequent data from the partner TP arrive at the local LU. (This method results in more efficient use of the network but requires a longer time to return control to the local TP.)

Use AP_SHORT to indicate that APPC returns control to the local TP when the confirmation from the partner TP arrives at the local LU.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_P_TO_R_INVALID_TYPE

Secondary return code; the **ptr_type** parameter was not set to a valid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_P_TO_R_NOT_SEND_STATE

Secondary return code; the conversation was not in SEND state.

AP_P_TO_R_NOT_LL_BDY

Secondary return code; the local TP did not finish sending a logical record.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it can indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued SEND_ERROR with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued a SEND_ERROR verb with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

Remarks

Before changing the conversation state, this verb performs the equivalent of one of the following:

- FLUSH, by sending the contents of the local LU's send buffer to the partner LU (and TP).

- CONFIRM, by sending the contents of the local LU's send buffer and a confirmation request to the partner TP.

After this verb has successfully executed, the local TP can receive data.

The conversation must be in SEND state when the TP issues this verb.

State changes, summarized in the following table, are based on the value of **primary_rc**.

primary_rc	**New state**
AP_OK	RECEIVE
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

The conversation does not change to SEND state for the partner TP until the partner TP receives one of the following values through the **what_rcvd** parameter of a subsequent receive verb:

- AP_SEND
- AP_CONFIRM_SEND and replies with CONFIRMED
- AP_DATA_COMPLETE_CONFIRM_SEND and replies with **CONFIRMED**
- AP_DATA_CONFIRM_SEND and replies with **CONFIRMED**

The receive verbs are RECEIVE_AND_POST, RECEIVE_IMMEDIATE, and RECEIVE_AND_WAIT.

RECEIVE_ALLOCATE

The **RECEIVE_ALLOCATE** verb is issued by the invoked transaction program (TP) to confirm that the invoked TP is ready to begin a conversation with the invoking TP that issued ALLOCATE or MC_ALLOCATE.

The following structure describes the verb control block (VCB) used by the **RECEIVE_ALLOCATE** verb.

Syntax

```
struct receive_allocate {
    unsigned short   opcode;
    unsigned char    opext;
    unsigned char    reserv2;
    unsigned short   primary_rc;
    unsigned long    secondary_rc;
    unsigned char    tp_name[64];
    unsigned char    tp_id[8];
    unsigned long    conv_id;
    unsigned char    sync_level;
    unsigned char    conv_type;
    unsigned char    user_id[10];
    unsigned char    lu_alias[8];
    unsigned char    plu_alias[8];
    unsigned char    mode_name[8];
    unsigned char    reserv3[2];
    unsigned long    conv_group_id;
    unsigned char    fqplu_name[17];
    unsigned char    pip_incoming;
    unsigned char    syncpoint_rqd;
    unsigned char    reserv4[3];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_RECEIVE_ALLOCATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_name

Supplied parameter. Provides the name of the local TP. The value of **tp_name** must match the TP name configured through registry or environment variables. APPC matches the **RECEIVE_ALLOCATE** verb's **tp_name** parameter with the TP name specified by the incoming allocate, which is generated by MC_ALLOCATE or ALLOCATE in the invoking TP.

This parameter is a 64-byte EBCDIC character string and is case-sensitive. The **tp_name** parameter can consist of characters from the type AE EBCDIC character set:

- Uppercase and lowercase letters
- Numerals 0 through 9

- Special characters $, #, and period (.)

If **tp_name** is fewer than 64 bytes, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention is that a service TP name can have up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the type AE EBCDIC character set.

tp_id

Returned parameter. Identifies the local TP.

conv_id

Returned parameter. Provides the conversation identifier. It identifies the conversation APPC has established between the two partner TPs.

sync_level

Returned parameter. Specifies the synchronization level of the conversation. It determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

- AP_NONE specifies that confirmation processing will not be used in this conversation.
- AP_CONFIRM_SYNC_LEVEL specifies that the TPs can use confirmation processing in this conversation.
- AP_SYNCPT specifies that TPs can use Sync Point Level 2 confirmation processing in this conversation.

conv_type

Returned parameter. Specifies the type of conversation chosen by the partner TP, using MC_ALLOCATE or ALLOCATE. The following are possible values:

AP_BASIC_CONVERSATION

AP_MAPPED_CONVERSATION

user_id

Returned parameter. Provides the user identifier specified by the partner TP, using **MC_ALLOCATE** or **ALLOCATE** (if the partner TP set the **MC_ALLOCATE** or **ALLOCATE** verb's security parameter to AP_PGM or AP_SAME). It is a type AE EBCDIC character string.

lu_alias

Returned parameter. Provides the alias by which the local logical unit (LU) is known to the local TP. It is an ASCII character string.

plu_alias

Returned parameter. Provides the alias by which the partner LU (which initiated the incoming allocate) is known to the local TP. It is an ASCII character string.

mode_name

Returned parameter. Provides the mode name specified by **MC_ALLOCATE** or **ALLOCATE** in the partner TP. It is the name of a set of networking characteristics defined during configuration. The **mode_name** is a type A EBCDIC character string.

reserv3

A reserved field.

conv_group_id

Conversation group identifier.

fqplu_name

This returned parameter provides the fully qualified LU name.

pip_incoming

This optional supplied and returned parameter is applicable only if Sync Point services are required.

For the supplied parameter:

AP_YES if TP does accept PIP data.

AP_NO if TP does not accept PIP data.

For the returned parameter:

AP_YES if PIP data is available.

AP_NO if PIP data is not available.

syncpoint_rqd

This parameter indicates if Sync Point services are required.

AP_YES if Sync Point is required.

AP_NO if Sync Point is not required.

reserv4

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_UNDEFINED_TP_NAME

Secondary return code; the TP name was not configured correctly.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_ALLOCATE_NOT_PENDING

Secondary return code; APPC did not find an incoming allocate (from the invoking TP) to match the value of **tp_name**, supplied by **RECEIVE_ALLOCATE**. **RECEIVE_ALLOCATE** waited for the incoming allocate and eventually timed out.

AP_INVALID_PROCESS

Secondary return code; the process issuing **RECEIVE_ALLOCATE** was different from the one started by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

This must be the first APPC verb issued by the invoked TP. The initial state is RESET. If the verb executes successfully (**primary_rc** is AP_OK), the state changes to RECEIVE.

In response to this verb, APPC establishes a conversation between the two TPs and generates a TP identifier for the invoked TP and a conversation identifier. These identifiers are required parameters for subsequent APPC verbs.

If the invoked TP issues **RECEIVE_ALLOCATE** and a corresponding incoming allocate (resulting from MC_ALLOCATE or ALLOCATE issued by the invoking TP) is not present, the invoked TP waits until the incoming allocate arrives or the verb times out. The time-out value is set by the system administrator.

Host Integration Server also supports APPC RECEIVE_ALLOCATE_EX and RECEIVE_ALLOCATE_EX_END functions, to simplify the design and implementation of some invokable transaction programs. This function allows an APPC application to receive all incoming FMH-5 Attach requests received by Host Integration Server over a specific Local APPC LU, allowing an application to act as an "attach manager." An attach manager is a program that handles an incoming FMH-5 Attach request to start an LU6.2 conversation. When an APPC application calls RECEIVE_ALLOCATE (as opposed to RECEIVE_ALLOCATE_EX), Host Integration Server handles the attach manager functionality.

For more information on attach manager, see RECEIVE_ALLOCATE_EX.

RECEIVE_ALLOCATE_EX

The RECEIVE_ALLOCATE_EX verb accepts a new VCB structure to allow registration of an attach manager.

Syntax

```
typedef struct receive_allocate_ex {
    unsigned short    opcode;
        unsigned char     opext;
        unsigned char     format;
        unsigned short    primary_rc;
        unsigned long    secondary_rc;
       unsigned char     tp_name[64];
       unsigned char     tp_id[8];
       unsigned long     conv_id;
       unsigned char     sync_level;
        unsigned char    conv_type;
        unsigned char     user_id[10];
       unsigned char     lu_alias[8];
       unsigned char     plu_alias[8];
       unsigned char     mode_name[8];
       unsigned char     reserv3[2];
       unsigned long     conv_group_id;
       unsigned char     fqplu_name[17];
       unsigned char     pip_incoming;
       unsigned long     timeout;
       unsigned char     password[10];
       unsigned char     reserv5[2];
       unsigned char     attach_id[8];
 }
```

Members

opcode

Supplied parameter: RECEIVE_ALLOCATE_EX

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

format

Reserved parameter.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued.

tp_name

Supplied parameter. The **tp_name** is a returned parameter only. However, the application must allocate sufficient buffer space to hold the **tp_name** (that is, 64 characters) and initialize the name to EBCDIC spaces (hexadecimal X'40')

The returned verb will contain the actual TP name sent by the remote system.

tp_id

Returned parameter. Identifies the local TP.

conv_id

Returned parameter. Provides the conversation identifier. It identifies the conversation APPC has established between the two partner TPs.

sync_level

Returned parameter. Specifies the synchronization level of the conversation. It determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

- **AP_NONE** specifies that confirmation processing will not be used in this conversation.
- **AP_CONFIRM_SYNC_LEVEL** specifies that the TPs can use confirmation processing in this conversation.

AP_SYNCPT specifies that TPs can use Sync Point Level 2 confirmation processing in this conversation.

conv_type

Returned parameter. Specifies the type of conversation chosen by the partner TP, using **MC_ALLOCATE** or **ALLOCATE**. The following are possible values:

AP_BASIC_CONVERSATION

AP_MAPPED_CONVERSATION

user_id

This is the EBCDIC user_id sent by the remote system

lu_alias

Supplied parameter. Local LU alias. Must be supplied to register an attach manager. Only one attach manager may be registered for a given Local LU alias within the Host Integration Server subdomain. If another attach manager process is already registered for this lu_alias, the following error will be returned:

primary_rc = AP_STATE_CHECK (0x0002) secondary_rc = AP_LU_ALREADY_REGISTERED (0x0000050A)

This indicates that Host Integration Server was unable to register this attach manager.

plu_alias

Returned parameter. Provides the alias by which the partner LU (which initiated the incoming allocate) is known to the local TP. It is an ASCII character string.

mode_name

Returned parameter. Provides the mode name specified by **MC_ALLOCATE** or **ALLOCATE** in the partner TP. It is the name of a set of networking characteristics defined during configuration. The mode_name is a type A EBCDIC character string.

reserv3

Reserved parameter.

conv_group_id

Conversation group identifier.

fqplu_name

This returned parameter provides the fully qualified LU name.

pip_incoming

Supplied parameter. If this attach manager will accept incoming FMH-5 Attaches which include PIP data, then set this to **AP_YES**. Otherwise, set this to **AP_NO**.

Returned parameter: If PIP data is present in the incoming Attach, this is set to **AP_YES**. If no PIP data is present, this will be set to **AP_NO**.

timeout

Timeout, in seconds. A value of 0xFFFFFFFF can be used to wait forever.

password

Returned parameter: This is the EBCDIC password sent by the remote system. If the remote system supports "Password substitution" (password encryption), the encrypted password will be received in **RECEIVE_ALLOCATE_EX**. There is no facility to decrypt this password, so the application will not be able to verify the user credentials.

reserv5

Reserved parameter.

attach_id

Returned parameter. Always set to 0. This field is defined for source compatibility with non-Microsoft SNA products.

Remarks

Host Integration Server supports APPC **RECEIVE_ALLOCATE_EX** and **RECEIVE_ALLOCATE_EX_END** to simplify the design and implementation of some invokable transaction programs. This function allows an APPC application to receive all incoming FMH-5 Attach requests received by Host Integration Server over a specific Local APPC LU, allowing an application to act as an "attach manager." An attach manager is a program that handles an incoming FMH-5 Attach request to start an LU6.2 conversation. When an APPC application calls **RECEIVE_ALLOCATE** (as opposed to **RECEIVE_ALLOCATE_EX**), Host Integration Server handles the attach manager functionality. To implement the attach manager functionality within an APPC application, the following occurs:

- The application supplies a Local APPC LU alias to the **RECEIVE_ALLOCATE_EX** function, with a tp_name of EBCDIC spaces (hexadecimal X'40').
- When Host Integration Server receives an incoming FMH-5 Attach request over an LU6.2 session using that Local APPC LU, Host Integration Server will route the request to the application.
- When the **RECEIVE_ALLOCATE_EX** completes, the application is responsible for the following:
    1. Accepting or rejecting the FMH-5 Attach
    2. Verifying any conversation level security, and
    3. If accepting the attach request, completely handling the request within its Win32 process context.
- To stop listening for new incoming attach requests, the application calls **RECEIVE_ALLOCATE_EX_END**.
- After a process issues **RECEIVE_ALLOCATE_EX**, the process should not call **RECEIVE_ALLOCATE** with a specific TP name. Likewise, if a process calls **RECEIVE_ALLOCATE**, that process should not later call **RECEIVE_ALLOCATE_EX**. In other words, for the duration of a process supporting an invokable TP, the process should exclusively call **RECEIVE_ALLOCATE**, or **RECEIVE_ALLOCATE_EX**, but not both.

It is not possible for the application to dispatch the incoming attach request to another process, as the conversation ID is only valid within its own application context.

Note
: Host Integration Server does not support auto starting an attach manager application. In other words, if an application calls **RECEIVE_ALLOCATE_EX**, the application must be started before any incoming Attach requests arrive over the Local LU.

The current specification does not return the Security Indicator (Byte 4 of the FMH-5 Attach). So, the application will need to accommodate incoming attach requests that contain the following:

1. Neither a user_id or password (when no security is sent in the attach),
2. A user_id only (such as for "already verified" attaches), or
3. Both a user_id and password (if user authorization is required).

In the LU6.2 BIND request, Host Integration Server indicates support for incoming FMH-5 Attach requests that contain user security, already verified, and password substitution. Host Integration Server does not support incoming attaches which request persistent verification.

The **RECEIVE_ALLOCATE_EX** function allows an application to register as an attach manager if the tp_name is set to all EBCDIC spaces (X'40') and a Local LU alias is supplied in the lu_alias field. When registered as an attach manager for a given lu_alias, Host Integration Server will route all incoming attaches received over the lu_alias to the application. See below for more information on how Host Integration Server routes incoming FMH-5 attach requests.

The application may call **RECEIVE_ALLOCATE_EX** more than once to register as the attach manager for one or more Local LU. However, only one attach manager may be registered on a given lu_alias within the SNA subdomain (that is, across all Host Integration Servers and attached Host Integration Server clients). The application cannot supply a blank tp_name and blank lu_alias. In other words, an application cannot register as the default attach manager to receive all incoming attach requests for an SNA subdomain.

When **RECEIVE_ALLOCATE_EX** completes, the application is responsible for the following:

- Deciding whether the attach will be accepted or not. Host Integration Server provides no mechanism for configuring transaction programs (TPs). The application must have its own means of defining which tp names it will support.

- If accepted, the application must verify conversation security attributes (user_id, password) and for handling the processing of the new conversation.

- The tp_id and conv_id cannot be passed to a separate process for handling. All TP processing must be provided by the application.

If the application chooses to reject the attach request, **[MC_]DEALLOCATE** must be called, specifying the conv_id received in the completed **RECEIVE_ALLOCATE_EX**, along with an appropriate reason code in the dealloc_type parameter, using these new extended codes:

```
#define AP_DEALLOC_SECURITY_NOT_VALID_PASSWORD_EXPIRED 0x10
#define AP_DEALLOC_SECURITY_NOT_VALID_PASSWORD_INVALID 0x11
#define AP_DEALLOC_SECURITY_NOT_VALID_USERID_REVOKED 0x12
#define AP_DEALLOC_SECURITY_NOT_VALID_USERID_INVALID 0x13
#define AP_DEALLOC_SECURITY_NOT_VALID_USERID_MISSING 0x14
#define AP_DEALLOC_SECURITY_NOT_VALID_PASSWORD_MISSING 0x15
#define AP_DEALLOC_SECURITY_NOT_VALID_GROUP_INVALID 0x16
#define AP_DEALLOC_SECURITY_NOT_VALID_USERID_REVOKED_IN_GROUP 0x17
#define AP_DEALLOC_SECURITY_NOT_VALID_USERID_NOT_DEFD_TO_GROUP 0x18
#define AP_DEALLOC_SECURITY_NOT_VALID_NOT_AUTHORIZED_AT_REMOTE_LU 0x19
#define AP_DEALLOC_SECURITY_NOT_VALID_NOT_AUTHORIZED_FROM_LOCAL_LU 0x1A
#define AP_DEALLOC_SECURITY_NOT_VALID_NOT_AUTHORIZED_TO_TRANSACTION_PROGRAM 0x1B
#define AP_DEALLOC_SECURITY_NOT_VALID_INSTALLATION_EXIT_FAILED 0x1C
#define AP_DEALLOC_SECURITY_NOT_VALID_PROCESSING_FAILURE 0x1D
#define AP_DEALLOC_SECURITY_NOT_VALID_PROTOCOL_VIOLATION 0x1E
```

When the application sets the above dealloc_type, the Host Integration Server then sends the corresponding sense code within the FMH-7 error sent to the remote system when rejecting the FMH-5 Attach request:

```
#define AP_SECURITY_NOT_VALID_PASSWORD_EXPIRED APPC_FLIPL(x080fff00)
#define AP_SECURITY_NOT_VALID_PASSWORD_INVALID APPC_FLIPL(x080fff01)
#define AP_SECURITY_NOT_VALID_USERID_REVOKED APPC_FLIPL(x080fff02)
#define AP_SECURITY_NOT_VALID_USERID_INVALID APPC_FLIPL(x080fff03)
#define AP_SECURITY_NOT_VALID_USERID_MISSING APPC_FLIPL(x080fff04)
#define AP_SECURITY_NOT_VALID_PASSWORD_MISSING APPC_FLIPL(x080fff05)
#define AP_SECURITY_NOT_VALID_GROUP_INVALID APPC_FLIPL(x080fff06)
#define AP_SECURITY_NOT_VALID_USERID_REVOKED_IN_GROUP APPC_FLIPL(x080fff07)
#define AP_SECURITY_NOT_VALID_USERID_NOT_DEFD_TO_GROUP APPC_FLIPL(x080fff08)
#define AP_SECURITY_NOT_VALID_NOT_AUTHORIZED_AT_REMOTE_LU APPC_FLIPL(x080fff09)
```

```
#define AP_SECURITY_NOT_VALID_NOT_AUTHORIZED_FROM_LOCAL_LU APPC_FLIPL(x080fff0A)

#define AP_SECURITY_NOT_VALID_NOT_AUTHORIZED_TO_TRANSACTION_PROGRAM APPC_FLIPL(x080fff0B)

#define AP_SECURITY_NOT_VALID_INSTALLATION_EXIT_FAILED APPC_FLIPL(x080fff0C)

#define AP_SECURITY_NOT_VALID_PROCESSING_FAILURE APPC_FLIPL(x080fff0D)

#define AP_SECURITY_NOT_VALID_PROTOCOL_VIOLATION APPC_FLIPL(x080fff0E)
```

Before calling **RECEIVE_ALLOCATE_EX**, the application may want to verify the configuration settings for the Local APPC LU to determine if the LU supports sync level 2, or is a member of the default LU pool. To do this, use the enhanced **GET_LU_STATUS** API.

To deregister as the attach manager for a given Local APPC LU, the application must call **RECEIVE_ALLOCATE_EX_END**, documented below. If an application has registered as the attach manager for more than one lu_alias, **RECEIVE_ALLOCATE_EX_END** must be called for each lu_alias.

The application should try to have a **RECEIVE_ALLOCATE_EX** pending at all times, in order to handle incoming attach requests in a timely manner. If the application fails to post new **RECEIVE_ALLOCATE_EX**, Host Integration Server queues up to 2,048 incoming attaches against the application, if the application is running on the Host Integration Server, or 256 if running on an SNA Windows NT client. If the limit is exceeded, Host Integration Server rejects the attach request with sense code X'084B6031', or **AP_TRANS_PGM_NOT_AVAIL_RETRY**.

RECEIVE_ALLOCATE_EX_END

The RECEIVE_ALLOCATE_EX_END verb allows an application to deregister as the attach manager for a given Local APPC LU (lu_alias). This verb must be called for each lu_alias previously passed to the RECEIVE_ALLOCATE_EX request.

Syntax

```
typedef struct receive_allocate_ex_end {
       unsigned short opcode;
       unsigned char reserv2[2];
       unsigned short primary_rc;
       unsigned long secondary_rc;
       unsigned char tp_name[64];
       unsigned char lu_alias[8];
       unsigned char reserved3[20];
 };
```

Members

Opcode

A supplied parameter. Specifies the verb operation code, RECEIVE_ALLOCATE_EX_END.

reserv2

A reserved field.

primary_rc

If the lu_alias has not been previously registered by the application, the following error is returned:

AP_STATE_CHECK (0x0002)

secondary_rc

If the lu_alias has not been previously registered by the application, the following error is returned:

AP_ATTACH_MANAGER_INACTIVE (0x00000508)

tp_name

Must be all EBCDIC spaces (X'40')

lu_alias

Must be supplied and must match the lu_alias provided in a previous RECEIVE_ALLOCATE_EX request from the same process

reserved3

A reserved field.

Remarks

If the application is providing sync point support, the application needs to know when the LU-LU session limits have dropped to zero. This can be done by polling the **GET_LU_STATUS** API.

After calling **RECEIVE_ALLOCATE_EX_END** to deregister an attach manager, Host Integration Server does not tear down any existing LU6.2 sessions. To tear down an existing session, call the **DEACTIVATE_SESSION** function, supplying the appropriate lu_alias and plu_alias. If you are using Sync Level 2, deactivating the LU6.2 sessions notifies the Remote LU that the syncpoint manager has gone away and thus, a new ExchangeLogNames is required for the next connection.

RECEIVE_AND_POST

The **RECEIVE_AND_POST** verb receives application data and status information asynchronously. This allows the local transaction program (TP) to proceed with processing while data is still arriving at the local logical unit (LU).

While an asynchronous **RECEIVE_AND_POST** is outstanding, the following verbs can be issued on the same conversation:

- DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
- GET_ATTRIBUTES
- GET_TYPE
- REQUEST_TO_SEND
- SEND_ERROR
- TEST_RTS
- TP_ENDED

This allows an application to use an asynchronous **RECEIVE_AND_POST** to receive data. While the **RECEIVE_AND_POST** is outstanding, it can still use **SEND_ERROR** and **REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support. For information on how a TP receives data and how to use this verb, see Remarks in this topic.

The following structure describes the verb control block (VCB) used by the **RECEIVE_AND_POST** verb.

Syntax

```
struct receive_and_post {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       fill;
    unsigned char       rts_rcvd;
    unsigned char       reserv4;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char FAR * sema;
    unsigned char       reserv5;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_RECEIVE_AND_POST.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or byRECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATEin the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Indicates whether data or conversation status was received. Possible values are listed following the Members section

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

- AP_NO specifies that indicators should be returned individually on separate invocations of the verb.
- AP_YES specifies that indicators should be returned together, provided both are available. Both can be returned when:

  The receive buffer is large enough to hold all of the data that precedes the status indicator.

  The **fill** parameter specifies BUFFER or LL, and the data is the last logical record before the status indicator.

fill

Supplied parameter. Specifies how the local TP receives data.

Use AP_BUFFER to indicate that the local TP receives data until the number of bytes specified by **max_len** is reached or until end of data. Data is received without regard for the logical-record format.

Use AP_LL to indicate that data is received in logical-record format. The data received can be:

- A complete logical record.
- A **max_len** byte portion of a logical record.
- The end of a logical record**.**

rts_rcvd

Returned parameter. Indicates whether the partner TP issued REQUEST_TO_SEND. Possible values are:

- AP_YES indicates that the partner TP issued **REQUEST_TO_SEND**, which requests that the local TP change the conversation to RECEIVE state.
- AP_NO indicates that the partner TP has not issued **REQUEST_TO_SEND**.

max_len

Supplied parameter. Specifies the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

The value must not exceed the length of the buffer to contain the received data. The offset of **dptr** plus the value of **max_len** must not exceed the size of the data segment.

Dlen

Returned parameter. Specifies the number of bytes of data received. Data is stored in the buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Provides the address of the buffer to contain the data received by the local LU.

For Microsoft® Windows® 2000, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

sema

Supplied parameter. Provides the address of the semaphore that APPC is to clear when the asynchronous receiving operation is finished. The **sema** parameter is an event handle obtained by calling either the **CreateEvent** or **OpenEvent** Win32 function.

Values Returned by the what_rcvd Parameter

- AP_CONFIRM_DEALLOCATE indicates that the partner TP issued DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL. The conversation's synchronization level, established by ALLOCATE, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues CONFIRMED.

- AP_CONFIRM_SEND indicates that the partner TP issued PREPARE_TO_RECEIVE with **ptr_type** set to AP_SYNC_LEVEL. The conversation's synchronization level, established by **ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **CONFIRMED**, and begins to send data.

- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP issued CONFIRM. Upon receiving this value, the local TP normally issues **CONFIRMED**.

- AP_DATA indicates that this value can be returned by **RECEIVE_AND_POST** if **fill** is set to AP_BUFFER. The local TP received data until **max_len** or the end of the data was reached. For more information, see Remarks in this topic.

- AP_DATA_COMPLETE indicates, for **RECEIVE_AND_POST**, that the local TP has received a complete data record or the last part of a data record.

  For **RECEIVE_AND_POST** with **fill** set to AP_LL, this value indicates that the local TP has received a complete logical record or the end of a logical record.

  Upon receiving this value, the local TP normally reissues **RECEIVE_AND_POST** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data.

  Otherwise, the local TP examines status information.

  If **primary_rc** contains AP_OK and **what_rcvd** contains AP_SEND, AP_CONFIRM_SEND, AP_CONFIRM_DEALLOCATE, or AP_CONFIRM_WHAT_RECEIVED, see the description of the value (in this section) for the next action the local TP normally takes.

  If **primary_rc** contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to the DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates, for **RECEIVE_AND_POST**, that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

  For **RECEIVE_AND_POST** with **fill** set to AP_LL, this value indicates that the local TP has received an incomplete logical record.

  Upon receiving this value, the local TP normally reissues **RECEIVE_AND_POST** (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.
- AP_SEND indicates, for the partner TP, that the conversation has entered RECEIVE state. For the local TP, the conversation is now in SEND state. Upon receiving this value, the local TP normally uses SEND_DATA to begin sending data.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_DATA_SEND

Primary return code; this is a combination of AP_DATA and AP_SEND.

AP_DATA_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_SEND.

AP_DATA_CONFIRM

Primary return code; this is a combination of AP_DATA and AP_CONFIRM.

AP_DATA_CONFIRM_DEALLOCATE

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE.

AP_DEALLOC_NORMAL

Primary return code; the partner TP issued DEALLOCATE with **dealloc_type** set to AP_FLUSH or AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE.

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_INVALID_SEMAPHORE_HANDLE

Secondary return code; the address of the RAM semaphore or system semaphore handle was invalid.

APPC cannot trap all invalid semaphore handles. If the TP passes a bad RAM semaphore handle, a protection violation results.

AP_RCV_AND_POST_BAD_FILL

Secondary return code; the **fill** parameter was set to an invalid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_AND_POST_BAD_STATE

Secondary return code; the conversation was not in RECEIVE or SEND state when the TP issued this verb.

AP_RCV_AND_POST_NOT_LL_BDY

Secondary return code; the conversation was in SEND state; the TP began but did not finish sending a logical record.

AP_CANCELED

Primary return code; the local TP issued one of the following verbs, which canceled **RECEIVE_AND_POST**:

DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER

SEND_ERROR

TP_ENDED

Issuing one of these verbs causes the semaphore to be cleared.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it can indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP issued SEND_ERROR with **err_type** set to AP_PROG while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_PROG_ERROR_TRUNC

Primary return code; in SEND state, after sending an incomplete logical record, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. The local TP may have received the first part of the logical record through a receive verb.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_NO_TRUNC

Primary return code; while in SEND state, the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC. Data was not truncated.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

AP_SVC_ERROR_TRUNC

Primary return code; in SEND state, after sending an incomplete logical record, the partner TP (or partner LU) issued **SEND_ERROR**. The local TP may have received the first part of the logical record.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received can be:

- One logical record.
- A buffer of data received independent of its logical-record format.

The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it. The receive verbs are **RECEIVE_AND_POST**, RECEIVE_AND_WAIT, and RECEIVE_IMMEDIATE.

2. The local TP issues the receive verb again. This has one of the following effects:

   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
   - If the partner TP has finished sending data or is waiting for confirmation, status information (available through **what_rcvd**) indicates the next action the local TP normally takes.

The following procedure shows tasks performed by the local TP in using **RECEIVE_AND_POST**.

To use RECEIVE_AND_POST

1. For the Microsoft Windows® 2000 operating system, the TP retrieves the WinAsyncAPPC message number by calling the **RegisterWindowMessage** API or allocating a semaphore. The **sema** field should be set to NULL if the application expects to be notified through the Windows message mechanism.

   APPC sends the Windows message or clears the semaphore when the local TP finishes receiving data.

   The semaphore will remain set while the local TP receives data asynchronously. APPC will clear the semaphore when the local TP finishes receiving data.

2. The TP issues **RECEIVE_AND_POST**.
3. The TP checks the value of **primary_rc**.

   If **primary_rc** is AP_OK, the receive buffer (pointed to by **dptr**) is asynchronously receiving data from the partner TP. While receiving data asynchronously, the local TP can:

   - Perform tasks not related to this conversation.
   - Issue REQUEST_TO_SEND.
   - Gather information about this conversation by issuing GET_TYPE, GET_ATTRIBUTES, or TEST_RTS.
   - Prematurely cancel **RECEIVE_AND_POST** by issuing DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER; SEND_ERROR;or TP_ENDED.

   If, however, **primary_rc** is not AP_OK, **RECEIVE_AND_POST** has failed. In this case, the local TP does not perform the next two tasks.

4. For the Windows 2000 operating system, when the TP finishes receiving data asynchronously, APPC issues the WinAsyncAPPC Windows message or clears the semaphore.
5. The TP checks the new value of **primary_rc**.

   If **primary_rc** is AP_OK, the local TP can examine the other returned parameters and manipulate the asynchronously received data.

   If **primary_rc** is not AP_OK, only **secondary_rc** and **rts_rcvd** (request-to-send received) are meaningful.

Conversation State Effects

The conversation must be in RECEIVE or SEND state when the TP issues this verb.

Issuing **RECEIVE_AND_POST** while the conversation is in SEND state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to PENDING_POST state; the local TP is ready to receive information from the partner TP asynchronously.

The conversation changes states twice:

- Upon initial return of the verb, if **primary_rc** contains AP_OK, the conversation changes to PENDING_POST state.
- After completion of the verb, the state changes depending on the value of the following:

  The **primary_rc** parameter

  The **what_rcvd** parameter if **primary_rc** is AP_OK

The following table shows the new state associated with each value of **what_rcvd** when **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	RECEIVE
AP_DATA_COMPLETE	RECEIVE
AP_DATA_INCOMPLETE	RECEIVE
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table shows the new state associated with each value of **primary_rc** other than AP_OK.

primary_rc	**New state**
AP_CANCELED	No change
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET

AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_PROG_ERROR_NO_TRUNC	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_NO_TRUNC	RECEIVE
AP_PROG_ERROR_TRUNC	RECEIVE
AP_SVC_ERROR_TRUNC	RECEIVE

End of Data for a Basic Conversation

If the local TP issues **RECEIVE_AND_POST** and sets **fill** to AP_BUFFER, the receipt of data ends when **max_len** or the end of the data is reached. The end of the data is indicated by either **primary_rc** with a value other than AP_OK (for example, AP_DEALLOC_NORMAL), or by **what_rcvd** with one of the following values:

AP_SEND

AP_CONFIRM_SEND

AP_CONFIRM_DEALLOCATE

AP_CONFIRM_WHAT_RECEIVED

AP_DATA_CONFIRM_SEND

AP_DATA_CONFIRM_DEALLOCATE

AP_DATA_CONFIRM

To determine if the end of the data has been reached, the local TP reissues **RECEIVE_AND_POST**. If the new **primary_rc** contains AP_OK and **what_rcvd** contains AP_DATA, the end of the data has not been reached. If, however, the end of the data has been reached, **primary_rc** or **what_rcvd** will indicate the cause of the end of the data.

Troubleshooting

The local TP can wait indefinitely if one of the following situations occurs:

- For the Windows 2000 operating system, the local TP issues a **RECEIVE_AND_POST** request, but either the partner TP has not sent data or the initial **primary_rc** is not AP_OK.

- For the OS/2 operating system, the local TP issues a **DosSemWait** function call, but either the partner TP has not sent data or the initial **primary_rc** is not AP_OK.

This is because APPC will not issue the Windows message or clear the semaphore.

When a condition resulting in one of the following **primary_rc** parameters occurs, APPC does not clear the semaphore:

AP_INVALID_SEMAPHORE_HANDLE

AP_INVALID_VERB_SEGMENT

AP_STACK_TOO_SMALL

To test **what_rcvd**, issue **RECEIVE_AND_POST** with **max_len** set to zero, so that the local TP can determine whether the partner TP has data to send, seeks confirmation, or has changed the conversation state.

RECEIVE_AND_WAIT

The **RECEIVE_AND_WAIT** verb receives any data that is currently available from the partner transaction program (TP). If no data is currently available, the local TP waits for data to arrive.

To allow full use to be made of the asynchronous support, asynchronously issued **RECEIVE_AND_WAIT** verbs have been altered to act like RECEIVE_AND_POST verbs. Specifically, while an asynchronous **RECEIVE_AND_WAIT** is outstanding, the following verbs can be issued on the same conversation:

- DEALLOCATE(AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
- GET_ATTRIBUTES
- GET_TYPE
- REQUEST_TO_SEND
- SEND_ERROR
- TEST_RTS
- TP_ENDED

This allows an application, and in particular, a 5250 emulator, to use an asynchronous **RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_WAIT** is outstanding, it can still use **SEND_ERROR** and **REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support.

The following structure describes the verb control block (VCB) used by the **RECEIVE_AND_WAIT** verb.

Syntax

```
struct receive_and_wait {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       fill;
    unsigned char       rts_rcvd;
    unsigned char       reserv4;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char       reserv5[5];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_RECEIVE_AND_WAIT.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Specifies the conversation identifier.

The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Indicates whether data or conversation status was received. Possible values are listed in the table following the Members section.

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

- AP_NO specifies that indicators should be returned individually on separate invocations of the verb.
- AP_YES specifies that indicators should be returned together, provided both are available. Both can be returned when:

  The receive buffer is large enough to hold all of the data that precedes the status indicator.

  The **fill** parameter specifies BUFFER or LL, and the data is the last logical record before the status indicator.

fill

Supplied parameter. Used in a basic conversation to specify how the local TP receives data. The following are allowed values:

- AP_BUFFER specifies that the local TP receives data until the number of bytes specified by **max_len** is reached or until the end of the data. Data is received without regard for the logical-record format.
- AP_LL specifies that data is received in logical-record format. The data received can be a complete logical record, a **max_len** byte portion of a logical record, or the end of a logical record.

rts_rcvd

Returned parameter. Contains the request-to-send indicator. Possible values are:

- AP_YES indicates that the partner TP has issued REQUEST_TO_SEND, which requests that the local TP change the conversation to RECEIVE state.
- AP_NO indicates that the partner TP has not issued **REQUEST_TO_SEND**.

max_len

Supplied parameter. Indicates the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

For the Microsoft Windows 2000 operating system, this value must not exceed the length of the buffer to contain the received data.

By issuing **RECEIVE_AND_WAIT** with **max_len** set to zero, the local TP can determine whether the partner TP has data to

send, seeks confirmation, or has changed the conversation state.

dlen

Returned parameter. Indicates the number of bytes of data received. Data is stored in the buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Provides the address of the buffer to contain the data received by the local TP.

For the Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Values Returned by the what_rcvd Parameter

- AP_CONFIRM_DEALLOCATE indicates that the partner TP has issued DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues CONFIRMED.

- AP_CONFIRM_SEND indicates that the partner TP has issued PREPARE_TO_RECEIVE with **ptr_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **CONFIRMED** and begins to send data.

- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP has issued CONFIRM. Upon receiving this value, the local TP normally issues **CONFIRMED**.

- AP_DATA can be returned in a basic conversation by **RECEIVE_AND_WAIT** if **fill** is set to AP_BUFFER. The local TP received data until **max_len** or end of data was reached. For more information, see "**RECEIVE_AND_WAIT** End of Data" at the end of this topic.

- AP_DATA_COMPLETE indicates, for **RECEIVE_AND_WAIT**, that the local TP has received a complete data record or the last part of a data record.

  For **RECEIVE_AND_WAIT** with **fill** set to AP_LL, this value indicates that the local TP has received a complete logical record or the end of a logical record.

  Upon receiving this value, the local TP normally reissues **RECEIVE_AND_WAIT** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data.

  Otherwise, the local TP examines status information, if **primary_rc** contains AP_OK and **what_rcvd** contains AP_SEND, AP_CONFIRM_SEND, AP_CONFIRM_DEALLOCATE, or AP_CONFIRM_WHAT_RECEIVED.

  See Return Codes in this topic for the next action the local TP normally takes.

  If **primary_rc** contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates, for **RECEIVE_AND_WAIT**, that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

  For **RECEIVE_AND_WAIT** with **fill** set to AP_LL, this value indicates that the local TP has received an incomplete logical record.

  Upon receiving this value, the local TP normally reissues **RECEIVE_AND_WAIT** (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.

- AP_SEND indicates, for the partner TP, that the conversation has entered RECEIVE state. For the local TP, the conversation

is now in SEND state. Upon receiving this value, the local TP normally uses SEND_DATA to begin sending data.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_DATA_SEND

Primary return code; this is a combination of AP_DATA and AP_SEND.

AP_DATA_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_SEND.

AP_DATA_CONFIRM

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_CONFIRM_DEALLOCATE

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation and issued DEALLOCATE with **dealloc_type** set to one of the following:

AP_CONFIRM_SYNC_LEVEL

AP_FLUSH

AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_RCV_AND_WAIT_BAD_FILL

Secondary return code; for basic conversations, **fill** was set to an invalid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_AND_WAIT_BAD_STATE

Secondary return code; the conversation was not in RECEIVE or SEND state when the TP issued this verb.

AP_RCV_AND_WAIT_NOT_LL_BDY

Secondary return code; for basic conversations, the conversation was in SEND state; the TP began but did not finish sending a logical record.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner logical unit (LU) or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued SEND_ERROR with **err_type** set to AP_PROG while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_PROG_ERROR_TRUNC

Primary return code; in SEND state, after sending an incomplete logical record, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. The local TP may have received the first part of the logical record through a receive verb.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP has issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_NO_TRUNC

Primary return code; while in SEND state, the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC. Data was not truncated.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

AP_SVC_ERROR_NO_TRUNC

Primary return code; while in SEND state, after sending an incomplete logical record, the partner TP (or partner LU) issued **SEND_ERROR**. The local TP may have received the first part of the logical record.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received can be:
   - One logical record.
   - A buffer of data received independent of its logical-record format.

   The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it.

   The receive verbs are RECEIVE_AND_POST, **RECEIVE_AND_WAIT**, and RECEIVE_IMMEDIATE.

2. The local TP issues the receive verb again. This has one of the following effects:

    - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
    - If the partner TP has finished sending data or is waiting for confirmation, status information (available through the **what_rcvd** parameter) indicates the next action the local TP normally takes.

The conversation must be in RECEIVE or SEND state when the TP issues this verb.

Issuing the Verb in SEND State

Issuing **RECEIVE_AND_WAIT** while the conversation is in SEND state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to RECEIVE state; the local TP waits for the partner TP to send data.

State Change

The new conversation state is determined by the following factors:

- The state the conversation is in when the TP issues the verb.
- The **primary_rc** parameter.
- The **what_rcvd** parameter if **primary_rc** contains AP_OK.

Verb Issued in SEND State

The following table details the state changes when **RECEIVE_AND_WAIT** is issued in SEND state and **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	RECEIVE
AP_DATA_COMPLETE	RECEIVE

AP_DATA_INCOMPLETE	RECEIVE
AP_SEND	No change
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when **RECEIVE_AND_WAIT** is issued in SEND state and **primary_rc** is not AP_OK.

primary_rc	**New state**
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_PROG_ERROR_NO_TRUNC	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_NO_TRUNC	RECEIVE

Verb Issued in RECEIVE State

The following table details the state changes when **RECEIVE_AND_WAIT** is issued in RECEIVE state and **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM

AP_DATA_CONFIRM	CONFIRM
AP_DATA	No change
AP_DATA_COMPLETE	No change
AP_DATA_INCOMPLETE	No change
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when **RECEIVE_AND_WAIT** is issued in RECEIVE state and **primary_rc** is not AP_OK.

primary_rc	**New state**
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	No change
AP_PROG_ERROR_NO_TRUNC	No change
AP_SVC_ERROR_PURGING	No change
AP_SVC_ERROR_NO_TRUNC	No change
AP_PROG_ERROR_TRUNC	No change
AP_SVC_ERROR_TRUNC	No change

RECEIVE_AND_WAIT End of Data

In basic conversations, if the local TP issues **RECEIVE_AND_WAIT** and sets **fill** to AP_BUFFER, the receipt of the data ends when **max_len** or the end of the data is reached. The end of the data is indicated by either:

- A **primary_rc** parameter with a value other than AP_OK (for example, AP_DEALLOC_NORMAL).
- A **what_rcvd** parameter with one of the following values:

  AP_SEND

  AP_CONFIRM_SEND

AP_CONFIRM_DEALLOCATE

AP_CONFIRM_WHAT_RECEIVED

AP_DATA_CONFIRM_SEND

AP_DATA_CONFIRM_DEALLOCATE

AP_DATA_CONFIRM

To determine if the end of the data has been reached, the local TP reissues **RECEIVE_AND_WAIT**. If the new **primary_rc** contains AP_OK and **what_rcvd** contains AP_DATA, the end of the data has not been reached. If, however, the end of the data has been reached, **primary_rc** or **what_rcvd** will indicate the cause of the end of the data.

RECEIVE_AND_WAIT waits for data or an indicator to be sent by the partner TP. If you need the local TP to operate continuously, use RECEIVE_IMMEDIATE instead.

RECEIVE_IMMEDIATE

The **RECEIVE_IMMEDIATE** verb receives any data currently available from the partner transaction program (TP). If no data is available, the local TP does not wait. To avoid blocking the conversation, the Microsoft Windows 2000 system can issue RECEIVE_AND_WAIT in conjunction with WinAsyncAPPC.

The following structure describes the verb control block used by the **RECEIVE_IMMEDIATE** verb.

Syntax

```
struct receive_immediate {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      what_rcvd;
    unsigned char       rtn_status;
    unsigned char       fill;
    unsigned char       rts_rcvd;
    unsigned char       reserv4;
    unsigned short      max_len;
    unsigned short      dlen;
    unsigned char FAR * dptr;
    unsigned char       reserv5[5];
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_RECEIVE_IMMEDIATE.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

what_rcvd

Returned parameter. Contains information received with the incoming data. Possible values are listed following the Members section.

rtn_status

Supplied parameter. Indicates whether both data and conversation status indicators should be returned within one API call.

Use AP_NO to specify that indicators should be returned individually on separate invocations of the verb.

Use AP_YES to specify that indicators should be returned together, provided both are available. Both can be returned when:

- The receive buffer is large enough to hold all of the data that precedes the status indicator.
- The **fill** parameter specifies either BUFFER or LL, and the data is the last logical record before the status indicator.

fill

Supplied parameter. Specifies the manner in which the local TP receives data. It is used only for **RECEIVE_IMMEDIATE**.

Use AP_BUFFER to indicate that the local TP receives data until the number of bytes specified by **max_len** is reached or until end of data. Data is received without regard for the logical-record format.

Use AP_LL to indicate that data is received in logical-record format. The data received can be a complete logical record, a **max_len** byte portion of a logical record, or the end of a logical record.

rts_rcvd

Returned parameter. Contains the request-to-send indicator. Possible values are:

- AP_YES indicates that the partner TP has issued REQUEST_TO_SEND, which requests that the local TP change the conversation to RECEIVE state.
- AP_NO indicates that the partner TP has not issued **REQUEST_TO_SEND**.

max_len

Supplied parameter. Indicates the maximum number of bytes of data the local TP can receive. The range is from 0 through 65535.

For the Windows 2000 operating system, this value must not exceed the length of the buffer to contain the received data.

By issuing **RECEIVE_IMMEDIATE** with **max_len** set to zero, the local TP can determine whether the partner TP has data to send, seeks confirmation, or has changed the conversation state.

dlen

Returned parameter. Provides the number of bytes of data received. Data is stored in a buffer specified by **dptr**. A length of zero indicates that no data was received.

dptr

Supplied parameter. Address of the buffer to contain the data received by the local TP.

For the Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Values Returned by the what_rcvd Parameter

- AP_CONFIRM_DEALLOCATE indicates that the partner TP has issued DEALLOCATE with **dealloc_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues CONFIRMED.
- AP_CONFIRM_SEND indicates that the partner TP has issued PREPARE_TO_RECEIVE with **ptr_type** set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by **ALLOCATE**, is AP_CONFIRM_SYNC_LEVEL. Upon receiving this value, the local TP normally issues **CONFIRMED**, and begins to send data.
- AP_CONFIRM_WHAT_RECEIVED indicates that the partner TP has issued CONFIRM. Upon receiving this value, the local TP normally issues **CONFIRMED**.
- AP_DATA is returned for basic conversations by **RECEIVE_IMMEDIATE** if **fill** is set to AP_BUFFER. The local TP received data until **max_len** or end of data was reached. For more information, see "RECEIVE_IMMEDIATE End of Data" at the end

of this topic.

- AP_DATA_COMPLETE indicates, for **RECEIVE_IMMEDIATE** with **fill** set to AP_LL in basic conversations, that the local TP has received a complete logical record or the end of a logical record.

  Upon receiving this value, the local TP normally reissues **RECEIVE_IMMEDIATE** or issues another receive verb. If the partner TP has sent more data, the local TP begins to receive a new unit of data.

  Otherwise, the local TP examines status information if **primary_rc** contains AP_OK and **what_rcvd** contains any of these values:

  AP_SEND

  AP_CONFIRM_SEND

  AP_CONFIRM_DEALLOCATE

  AP_CONFIRM_WHAT_RECEIVED

  See the description of the value in Return Codes in this topic for the next action the local TP normally takes.

  If **primary_rc** contains AP_DEALLOC_NORMAL, the conversation has been deallocated in response to DEALLOCATE issued by the partner TP.

- AP_DATA_INCOMPLETE indicates for **RECEIVE_IMMEDIATE** in mapped conversations that the local TP has received an incomplete data record. The **max_len** parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

  For **RECEIVE_IMMEDIATE** with **fill** set to AP_LL in basic conversations, this value indicates that the local TP has received an incomplete logical record.

  Upon receiving this value, the local TP normally reissues RECEIVE_IMMEDIATE (or issues another receive verb) to receive the next part of the record.

- AP_NONE indicates that the TP did not receive data or conversation status indicators.

- AP_SEND indicates, for the partner TP, the conversation has entered RECEIVE state. For the local TP, the conversation is now in SEND state. Upon receiving this value, the local TP normally uses SEND_DATAto begin sending data.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

When **rtn_status** is AP_YES, the preceding return code or one of the following return codes can be returned.

AP_DATA_COMPLETE_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_SEND.

AP_DATA_COMPLETE_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND.

AP_DATA_COMPLETE_CONFIRM

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_COMPLETE_CONFIRM_DEALL

Primary return code; this is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE.

AP_DATA_SEND

Primary return code; this is a combination of AP_DATA and AP_SEND.

AP_DATA_CONFIRM_SEND

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_SEND.

AP_DATA_CONFIRM

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_WHAT_RECEIVED.

AP_DATA_CONFIRM_DEALLOCATE

Primary return code; this is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE.

AP_UNSUCCESSFUL

Primary return code; no data is immediately available from the partner TP.

AP_DEALLOC_NORMAL

Primary return code; the partner TP has deallocated the conversation without requesting confirmation. The partner TP issued DEALLOCATE with **dealloc_type** set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

If **rtn_status** is AP_YES, examine **what_rcvd** also.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA

Secondary return code; the specified **rtn_status** value was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_RCV_IMMD_BAD_FILL

Secondary return code for a basic conversation; the **fill** parameter was set to an invalid value.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_RCV_IMMD_BAD_STATE

Secondary return code; the conversation was not in RECEIVE state.

AP_ALLOCATION_ERROR

Secondary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner logical unit (LU) or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it can indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_NO_TRUNC

Primary return code; the partner TP has issued SEND_ERROR with **err_type** set to AP_PROG while the conversation was in SEND state. Data was not truncated.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_PROG_ERROR_TRUNC

Primary return code; in SEND state, after sending an incomplete logical record, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. The local TP may have received the first part of the logical record through a receive verb.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP has issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_NO_TRUNC

Primary return code; while in SEND state, the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC. Data was not truncated.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

AP_SVC_ERROR_TRUNC

Primary return code; in SEND state, after sending an incomplete logical record, the partner TP (or partner LU) issued **SEND_ERROR**. The local TP may have received the first part of the logical record.

Remarks

The local TP receives data through the following process:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received can be:

   - One logical record.
   - A buffer of data received independent of its logical-record format.

   The local TP may need to issue the receive verb several times in order to receive a complete unit of data. After a complete unit of data has been received, the local TP can manipulate it.

   The receive verbs are RECEIVE_AND_POST, RECEIVE_AND_WAIT, and **RECEIVE_IMMEDIATE**.

2. The local TP issues the receive verb again. This has one of the following effects:

   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.
   - If the partner TP has finished sending data or is waiting for confirmation, status information (available through **what_rcvd**) indicates the next action the local TP normally takes.

The conversation must be in RECEIVE state when the TP issues this verb.

The new state is determined by **primary_rc**. If **primary_rc** is AP_OK, the new state is determined by **what_rcvd**.

The following table details the state changes when the **primary_rc** is AP_OK.

what_rcvd	**New state**
AP_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL	CONFIRM_DEALLOCATE
AP_DATA_CONFIRM_DEALLOCATE	CONFIRM_DEALLOCATE
AP_CONFIRM_SEND	CONFIRM_SEND

AP_DATA_COMPLETE_CONFIRM_SEND	CONFIRM_SEND
AP_DATA_CONFIRM_SEND	CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED	CONFIRM
AP_DATA_COMPLETE_CONFIRM	CONFIRM
AP_DATA_CONFIRM	CONFIRM
AP_DATA	No change
AP_DATA_COMPLETE	No change
AP_DATA_INCOMPLETE	No change
AP_SEND	SEND
AP_DATA_COMPLETE_SEND	SEND_PENDING

The following table details the state changes when the **primary_rc** is not AP_OK.

primary_rc	**New state**
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	No change
AP_PROG_ERROR_NO_TRUNC	No change
AP_SVC_ERROR_PURGING	No change
AP_SVC_ERROR_NO_TRUNC	No change
AP_PROG_ERROR_TRUNC	No change
AP_SVC_ERROR_TRUNC	No change
AP_UNSUCCESSFUL	No change

RECEIVE IMMEDIATE End of Data

In basic conversations, if the local TP issues **RECEIVE_IMMEDIATE** and sets **fill** to AP_BUFFER, the receipt of the data ends when **max_len** or the end of the data is reached. The end of the data is indicated by either:

- A **primary_rc** parameter with a value other than AP_OK (for example, AP_DEALLOC_NORMAL).
- A **what_rcvd** parameter with one of the following values:

  AP_SEND

  AP_CONFIRM_SEND

  AP_CONFIRM_DEALLOCATE

  AP_CONFIRM_WHAT_RECEIVED

  AP_DATA_CONFIRM_SEND

  AP_DATA_CONFIRM_DEALLOCATE

  AP_DATA_CONFIRM

To determine if the end of the data has been reached, the local TP reissues **RECEIVE_IMMEDIATE**. If the new **primary_rc** parameter contains AP_OK and **what_rcvd** contains AP_DATA, the end of the data has not been reached. If, however, the end of the data has been reached, **primary_rc** or **what_rcvd** will indicate the cause of the end of the data.

RECEIVE_LOG_DATA

The **RECEIVE_LOG_DATA** verb allows the user to register to receive the log data associated with an inbound Function Management Header 7 (FMH7) error report. The verb passes a buffer to APPC, and any log data received is placed in that buffer. APPC continues to use this buffer as successive FMH7s arrive until it is provided with another one (that is, until the transaction program (TP) issues another **RECEIVE_LOG_DATA** specifying a different buffer or no buffer at all).

Note that the TP itself is responsible for allocating and freeing the buffer. After the buffer has been passed to APPC, the TP should either issue another **RECEIVE_LOG_DATA** specifying a new buffer or a zero-length buffer, or wait until the conversation has finished before freeing the original buffer.

When an FMH7 is received, APPC copies any associated error log general data stream (GDS) into the buffer. If there is no associated error log variable, the buffer is zeroed out. It is up to the TP to check the buffer whenever a return code from a receive verb indicates that an error has been received.

The following structure describes the verb control block (VCB) used by the **RECEIVE_LOG_DATA** verb.

Syntax

```
struct receive_log_data {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv1;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned short      log_dlen;
    unsigned char FAR * log_dptr;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_RECEIVE_LOG_DATA.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv1

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

log_dlen

Supplied parameter. Specifies the maximum length of log data that APPC can place in the buffer (that is, the buffer size). The range is from 0 through 65535. Note that a length of zero here indicates that any previous **RECEIVE_LOG_DATA** verb

should be cancelled.

log_dptr

Supplied parameter. Specifies the address of the buffer that APPC will use to store the log data.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

REQUEST_TO_SEND

The **REQUEST_TO_SEND** verb notifies the partner transaction program (TP) that the local TP wants to send data.

The following structure describes the verb control block (VCB) used by the **REQUEST_TO_SEND** verb.

Syntax

```
struct request_to_send {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_REQUEST_TO_SEND.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_R_T_S_BAD_STATE

Secondary return code; the conversation is not in an allowed state when the TP issued this verb.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Microsoft Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any of the following states when the TP issues this verb:

CONFIRM

PENDING_POST (OS/2)

RECEIVE

There is no state change.

The request-to-send notification is received by the partner program through the **rts_rcvd** parameter of the following verbs:

- CONFIRM
- RECEIVE_AND_POST
- RECEIVE_AND_WAIT
- RECEIVE_IMMEDIATE
- SEND_DATA
- SEND_ERROR

It is also indicated by a **primary_rc** of AP_OK on TEST_RTS.

Request-to-send notification is sent to the partner TP immediately; APPC does not wait until the send buffer fills up or is flushed. Consequently, the request-to-send notification may arrive out of sequence. For example, if the local TP is in SEND state and issues PREPARE_TO_RECEIVE followed by **REQUEST_TO_SEND**, the partner TP, in RECEIVE state, may receive the request-to-send notification before it receives the send notification. For this reason, request-to-send can be reported to a TP through a receive verb.

In response to this request, the partner TP can change the conversation to:

- RECEIVE state by issuing **PREPARE_TO_RECEIVE** or **RECEIVE_AND_WAIT**.
- PENDING_POST state by issuing **RECEIVE_AND_POST**.

The partner TP can also ignore the request-to-send.

The conversation state changes to SEND for the local TP when the local TP receives one of the following values through the **what_rcvd** parameter of a subsequent receive verb:

- AP_CONFIRM_SEND and replies with CONFIRMED
- AP_DATA_COMPLETE_CONFIRM_SEND and replies with **CONFIRMED**
- AP_DATA_CONFIRM_SEND and replies with **CONFIRMED**
- AP_SEND

The receive verbs are RECEIVE_AND_POST, RECEIVE_IMMEDIATE, and RECEIVE_AND_WAIT.

SEND_CONVERSATION

The **SEND_CONVERSATION** verb allocates a session between the local logical unit (LU) and partner LU, sends data on the session, and then deallocates the session.

The following structure describes the verb control block (VCB) used by the **SEND_CONVERSATION** verb.

Syntax

```
struct send_conversation {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
       unsigned char       reserv3[8];
    unsigned char       rtn_ctl;
    unsigned char       reserv4;
    unsigned long       conv_group_id;
    unsigned long       sense_data;
    unsigned char       plu_alias[8];
    unsigned char       mode_name[8];
    unsigned char       tp_name[64];
    unsigned char       security;
    unsigned char       reserv6[11];
    unsigned char       pwd[10];
    unsigned char       user_id[10];
    unsigned short      pip_dlen;
    unsigned char FAR * pip_dptr;
    unsigned char       reserv6;
    unsigned char       fqplu_name[17];
    unsigned char       reserv7[8];
    unsigned short      dlen;
    unsigned char FAR * dptr;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_SEND_CONVERSATION.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local transaction program (TP). The value of this parameter was returned by TP_STARTED.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by ALLOCATE in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

rtn_ctl

Supplied parameter. Specifies how APPC should select a session to allocate for the conversation and when the local LU should return control to the local TP. The allowed values are:

- AP_IMMEDIATE specifies that the LU allocates a contention-winner session, if one is immediately available, and returns control to the TP.
- AP_WHEN_SESSION_ALLOCATED specifies that the LU does not return control to the TP until it allocates a session or encounters one of the errors described in Return Codes in this topic. If the session limit is zero, the LU returns control immediately. Note that if a session is not available, the TP waits for one.
- AP_WHEN_SESSION_FREE specifies that the LU allocates a contention-winner or contention-loser session, if one is available or able to be activated, and returns control to the TP. If an error occurs (as described in Return Codes in this topic) the call will return immediately with the error in the **primary_rc** and **secondary_rc** fields.
- AP_WHEN_CONWINNER_ALLOC specifies that the LU does not return control until it allocates a contention-winner session or encounters one of the errors described in Return Codes in this topic. If the session limit is zero, the LU returns control immediately. Note that if a session is not available, the TP waits for one.
- AP_WHEN_CONV_GROUP_ALLOC specifies that the LU does not return control to the TP until it allocates the session specified by **conv_group_id** or encounters one of the errors described in Return Codes in this topic. If the session is not available, the TP waits for it to become free.

conv_group_id

Supplied/returned parameter. Used as a supplied parameter when **rtn_ctl** is WHEN_CONV_GROUP_ALLOC to specify the identity of the conversation group from which the session should be allocated. When **rtn_ctl** specifies a different value, and the **primary_rc** is AP_OK, this is a returned value. The purpose of this parameter is to provide a TP with the assurance that the same session will be reallocated and therefore the conversations conducted over the session will occur in the same sequence that they were initiated.

sense_data

Returned parameter. If the primary and secondary return codes indicate an allocation error (retry or no-retry), an SNA-defined sense code is returned.

plu_alias

Supplied parameter. Specifies the alias by which the partner LU is known to the local TP. This parameter must match the name of a partner LU established during configuration. The parameter is an 8-byte, type G ASCII character set that includes:

- Uppercase letters
- Numerals 0 to 9
- Spaces
- Special characters $, #, %, and @

If the value of this parameter is fewer than eight bytes, pad it on the right with ASCII spaces (0x20).

mode_name

Supplied parameter. Specifies the name of a set of networking characteristics defined during configuration. This parameter must match the name of a mode associated with the partner LU during configuration.

The parameter is an 8-byte EBCDIC character string. It can consist of characters from the type A EBCDIC character set, including all EBCDIC spaces. These characters are:

- Uppercase letters

- Numerals 0 to 9
- Special characters $, #, and @

The first character in the string must be an uppercase letter or special character.

Using the name SNASVCMG (a reserved mode name used internally by APPC) in a basic conversation is not recommended.

tp_name

Supplied parameter. Specifies the name of the invoked TP. The value of **tp_name** specified by ALLOCATE in the invoking TP must match the value of **tp_name** specified by RECEIVE_ALLOCATE in the invoked TP.

The parameter is a 64-byte, case-sensitive, EBCDIC character string. This parameter can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC spaces (0x40) to pad it on the right.

The SNA convention for naming a service TP is up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the EBCDIC AE character set.

security

Supplied parameter. Specifies the information the partner LU requires in order to validate access to the invoked TP.

- AP_NONE specifies that the invoked TP uses no conversation security.
- AP_PGM specifies that the invoked TP uses conversation security and requires a user identifier and password. Use **user_id** and **pwd** to supply this information.
- AP_SAME specifies that the invoked TP, invoked with a valid user identifier and password, in turn invokes another TP.

For example, assume that TP A invokes TP B with a valid user identifier and password, and TP B in turn invokes TP C. If TP B specifies the value AP_SAME, APPC will send the LU for TP C the user identifier from TP A and an already-verified indicator. This indicator indicates to TP C not to require the password (if TP C is configured to accept an already-verified indicator).

pwd

Supplied parameter. Specifies the password associated with **user_id**. This parameter is required only if the security parameter is set to AP_PGM and must match the password for **user_id** that was established during configuration.

This parameter is a 10-byte, case-sensitive, EBCDIC character string. It can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

user_id

Supplied parameter. Specifies the user identifier required to access the partner TP. This parameter is required only if the security parameter is set to AP_PGM and must match one of the user identifiers configured for the partner TP.

The parameter can consist of characters from the type AE EBCDIC character set. These characters are:

- Uppercase and lowercase letters
- Numerals 0 to 9
- Special characters $, #, @, and period (.)

If the user identifier is fewer than 10 bytes, use EBCDIC spaces (0x40) to pad it on the right.

pip_dlen

Supplied parameter. Specifies the length of the PIP to be passed to the partner TP. The range for this parameter is from 0 through 32767.

pip_dptr

Supplied parameter. Specifies the address of the buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

PIP data can consist of initialization parameters or environmental setup information required by a partner TP or remote operating system. The PIP data must follow the GDS format. For more information, see your IBM SNA manual(s).

For the Microsoft Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area.

fqplu_name

Supplied parameter. Specifies the fully qualified name of the local LU. This parameter must match the fully qualified name of the local LU defined in the remote node. The parameter is made up of two type A EBCDIC character strings (each of up to eight characters), which are the network name (NETID) and the LU name of the partner LU. The names are separated by an EBCDIC period (.). The NETID can be omitted, and if this is the case, the period should also be omitted.

This name must be provided if no **plu_alias** is provided.

Type A EBCDIC characters contain:

- Uppercase letters
- Numerals 0 to 9
- Special characters $, #, and @

If the value of this parameter is fewer than 17 bytes, pad it on the right with EBCDIC spaces (0x40).

dlen

Supplied parameter. Specifies the number of bytes of data to be put in the local LU's send buffer. The range for this parameter is from 0 through 65535.

dptr

Supplied parameter. Specifies the address of the buffer containing the data to be put in the local LU's send buffer.

For the Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; the supplied parameter **rtn_ctl** specified immediate return of the control to the TP (AP_IMMEDIATE), and the local LU did not have an available contention-winner session.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_RETURN_CONTROL

Secondary return code; the value specified for **rtn_ctl** was invalid.

AP_BAD_SECURITY

Secondary return code; the value specified for **security** was invalid.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_PIP_LEN_INCORRECT

Secondary return code; the value of **pip_dlen** was greater than 32767.

AP_UNKNOWN_PARTNER_MODE

Secondary return code; the value specified for **mode_name** was invalid.

AP_BAD_PARTNER_LU_ALIAS

Secondary return code; APPC did not recognize the supplied **partner_lu_alias**.

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with ALLOCATE, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with TP_STARTED is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

This verb is issued by the invoking TP to conduct an entire conversation with the remote TP. If the remote TP rejects either the conversation initiation or the data, the invoking TP will not receive notification of the rejection.

The conversation state is RESET when the TP issues this verb. There is no state change.

Several parameters of **SEND_CONVERSATION** are EBCDIC or ASCII strings. A TP can use the CSV CONVERT to translate a string from one character set to the other.

Normally, the value of **mode_name** must match the name of a mode configured for the invoked TP's node and associated during configuration with the partner LU. If one of the modes associated with the partner LU on the invoked TP's node is an implicit mode, the session established between the two LUs will be of the implicit mode when no mode name associated with the partner LU matches the value of **mode_name**.

SEND_DATA

The **SEND_DATA** verb places data in the local logical unit's (LU) send buffer for transmission to the partner transaction program (TP).

The following structure describes the verb control block (VCB) used by the **SEND_DATA** verb.

Syntax

```
struct send_data {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       rts_rcvd;
       unsigned char       data_type;
    unsigned short int  dlen;
    unsigned char FAR * dptr ;
    unsigned char       type;
    unsigned char       reserv4;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_SEND_DATA.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier.

The value of this parameter is returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Provides the request-to-send-received indicator.

- AP_YES indicates that the partner TP has issued REQUEST_TO_SEND, which requests that the local TP change the conversation to RECEIVE state. To change to RECEIVE state, the local TP can use PREPARE_TO_RECEIVE, RECEIVE_AND_WAIT, or RECEIVE_AND_POST.

- AP_NO indicates that the partner TP has not issued **REQUEST_TO_SEND**.

data_type

Supplied parameter. Specifies the type of data to be sent if Sync Point is supported. Valid parameters are:

AP_APPLICATION

AP_USER_CONTROL_DATA

AP_PS_HEADER

dlen

Supplied parameter. Specifies the number of bytes of data to be put in the local LU's send buffer. The range is from 0 through 65535.

dptr

Supplied parameter. Specifies the address of the buffer containing the data to be put in the local LU's send buffer.

For the Microsoft Windows 2000 operating system, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

type

Supplied parameter. Allows a TP to send data and perform other functions within one API call. For example, you can combine **SEND_DATA** with **type** set to CONFIRM to accomplish the same objective as issuing **SEND_DATA** followed by CONFIRM.

- AP_SEND_DATA_CONFIRM corresponds to **SEND_DATA** followed by **CONFIRM**.
- AP_SEND_DATA_FLUSH corresponds to **SEND_DATA** followed by FLUSH.
- AP_SEND_DATA_DEALLOC_ABEND corresponds to **SEND_DATA** followed by DEALLOCATE with a **dealloc_type** of AP_ABEND_PROG.
- AP_SEND_DATA_DEALLOC_FLUSH corresponds to **SEND_DATA** followed by **DEALLOCATE** with a **dealloc_type** of AP_FLUSH.
- AP_SEND_DATA_DEALLOC_SYNC_LEVEL corresponds to **SEND_DATA** followed by **DEALLOCATE** with a **dealloc_type** of AP_SYNC_LEVEL.
- AP_SEND_DATA_P_TO_R_FLUSH corresponds to **SEND_DATA** followed by PREPARE_TO_RECEIVE with a **ptr_type** of AP_FLUSH.
- AP_SEND_DATA_P_TO_R_SYNC_LEVEL corresponds to **SEND_DATA** followed by **PREPARE_TO_RECEIVE** with a **ptr_type** of AP_SYNC_LEVEL and **locks** set to AP_SHORT.

reserv4

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_LL

Secondary return code; the logical record length field of a logical record contained an invalid value—0x0000, 0x0001,

0x8000, or 0x8001. See About Transaction Programs for information on logical records.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the length specified for the data buffer was longer than the segment allocated to contain the buffer.

AP_SEND_DATA_INVALID_TYPE

Secondary return code; the specified type was not recognized by APPC.

AP_SEND_DATA_CONFIRM_SYNC_NONE

Secondary return code; the **type** CONFIRM is not permitted for a conversation that was allocated with a **sync_level** of NONE.

AP_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

AP_SEND_DATA_NOT_SEND_STATE

Secondary return code; the local TP issued **SEND_DATA**, but the conversation was not in SEND state.

AP_SEND_DATA_NOT_LL_BDY

Secondary return code; the TP started but did not finish sending a logical record. This occurs only when the **type** parameter is one of the following:

AP_SEND_DATA_CONFIRM

AP_SEND_DATA_DEALLOC_FLUSH

AP_SEND_DATA_DEALLOC_SYNC_LEVEL

AP_SEND_DATA_P_TO_R_FLUSH

AP_SEND_DATA_P_TO_R_SYNC_LEVEL

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **ALLOCATE**, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with **TP_STARTED** is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. Data sent but not yet received is purged.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP has issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued a **SEND_ERROR** verb with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

Remarks

The conversation must be in SEND state when the TP issues this verb. State changes, based on **primary_rc**, are summarized in the following table.

primary_rc	**New state**
AP_OK	No change
AP_ALLOCATION_ERROR	RESET

AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

SEND_DATA may wait indefinitely because the partner TP has not issued a receive verb. If this occurs, the send buffer may fill up.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) when one of the following occurs:

- The send buffer fills up.
- The local TP issues FLUSH, CONFIRM, or DEALLOCATE (or other verb that flushes the LU's send buffer).

SEND_ERROR

The **SEND_ERROR** verb notifies the partner transaction program (TP) that the local TP has encountered an application-level error.

The following structure describes the verb control block (VCB) used by the **SEND_ERROR** verb.

Syntax

```
struct send_error {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       rts_rcvd;
    unsigned char       err_type;
    unsigned char       err_dir;
    unsigned char       reserv3;
    unsigned short      log_dlen;
    unsigned char FAR * log_dptr;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_SEND_ERROR.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP.

The value of this parameter is returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter is returned by ALLOCATEin the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

rts_rcvd

Returned parameter. Indicates whether the partner TP issued REQUEST_TO_SEND.Possible values include:

- AP_YES indicates that the partner TP has issued **REQUEST_TO_SEND**, which requests that the local TP change the conversation to RECEIVE state. To change to RECEIVE state, the local TP can use PREPARE_TO_RECEIVE, RECEIVE_AND_WAIT, or RECEIVE_AND_POST.

- AP_NO indicates that the partner TP has not issued **REQUEST_TO_SEND**.

err_type

Supplied parameter. Indicates the type of the error being reported—application program or service program.

AP_PROG indicates that the error is to be reported to an end-user application program. This value causes APPC to send one of the following return codes to the partner TP:

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING

AP_PROG_ERROR_TRUNC

AP_SVC indicates that the error is to be reported to a service program. This value causes APPC to send one of the following return codes to the partner TP:

AP_SVC_ERROR_NO_TRUNC

AP_SVC_ERROR_PURGING

AP_SVC_ERROR_TRUNC

err_dir

Supplied parameter. Indicates whether the error is with data just received or with data that is about to be sent. Use this parameter only when the conversation is in SEND_PENDING state. The parameter is ignored otherwise. The following are allowed values:

- AP_RCV_DIR_ERROR indicates that the TP issued **SEND_ERROR** after detecting an error associated with the data just received.
- AP_SEND_DIR_ERROR indicates that the TP issued **SEND_ERROR** after detecting an error associated with data it was going to send. For example, the TP encountered an error while reading data from the disk drive.

reserv3

A reserved field.

log_dlen

Supplied parameter for basic conversations; specifies the number of bytes of data to be sent to the error log file. The range is from 0 through 32767.

A length of zero indicates that there is no error log data.

log_dptr

Supplied parameter for basic conversations; specifies the address of the data buffer containing error information. The data is sent to the local error log and to the partner logical unit (LU).

This parameter is used by **SEND_ERROR** if **log_dlen** is greater than zero.

For Microsoft Windows 2000, the data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

The TP must format the error data as a GDS error log variable. For more information, see your IBM SNA manual(s).

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_BAD_ERROR_DIRECTION

Secondary return code; the specified **err_dir** was not recognized by APPC.

AP_INVALID_DATA_SEGMENT

Secondary return code; the error data for the log file was longer than the segment allocated to contain the error data, or the address of the error data buffer was wrong.

AP_SEND_ERROR_BAD_TYPE

Secondary return code; the value of **err_type** was invalid.

AP_SEND_ERROR_LOG_LL_WRONG

Secondary return code; the LL field of the error log GDS variable did not match the actual length of the data.

The following return codes can be generated when **SEND_ERROR** is issued in any allowed state:

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **ALLOCATE**, it may indicate that no communications system could be found to support the local LU. (For example, the local LU alias specified with **TP_STARTED** is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONV_FAILURE_NO_RETRY

Primary return code; the conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY

Primary return code; the conversation was terminated because of a temporary error. Restart the TP to see if the problem

occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This may occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

The following return codes can be generated only if **SEND_ERROR** is issued in SEND state:

AP_ALLOCATION_ERROR

Primary return code; APPC has failed to allocate a conversation. The conversation state is set to RESET.

This code may be returned through a verb issued after ALLOCATE.

AP_ALLOCATION_FAILURE_NO_RETRY

Secondary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY

Secondary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

AP_CONVERSATION_TYPE_MISMATCH

Secondary return code; the partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED

Secondary return code; the allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

AP_PIP_NOT_SPECIFIED_CORRECTLY

Secondary return code; the partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID

Secondary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED

Secondary return code; the partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

AP_TP_NAME_NOT_RECOGNIZED

Secondary return code; the partner LU does not recognize the TP name specified in the allocation request.

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

AP_TRANS_PGM_NOT_AVAIL_RETRY

Secondary return code; the remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

AP_PROG_ERROR_PURGING

Primary return code; while in RECEIVE, PENDING, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the partner TP issued **SEND_ERROR** with **err_type** set to AP_PROG. Data sent but not yet received is purged.

The following return codes can be generated only if **SEND_ERROR** is issued in SEND state:

AP_DEALLOC_ABEND_PROG

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner TP has issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP has encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner TP issued **DEALLOCATE** with **dealloc_type** set to AP_ABEND_TIMER.

AP_SVC_ERROR_PURGING

Primary return code; the partner TP (or partner LU) issued **SEND_ERROR** with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

The following return code can be generated only if **SEND_ERROR** is issued in RECEIVE state:

AP_DEALLOC_NORMAL

Primary return code; this return code does not indicate an error.

The partner TP issued DEALLOCATE with **dealloc_type** set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

Remarks

The conversation can be in any state except RESET when the TP issues this verb. The conversation state must be SEND_PENDING if **err_dir** is used.

The local TP sends the error notification immediately to the partner TP; it does not hold the information in the local LU's send buffer.

Upon successful execution of this verb, the conversation is in SEND state for the local TP and in RECEIVE state for the partner TP.

The new state is determined by **primary_rc**. Possible state changes are summarized in the following table.

primary_rc	New state
AP_OK	SEND
AP_ALLOCATION_ERROR	RESET
AP_CONV_FAILURE_RETRY	RESET
AP_CONV_FAILURE_NO_RETRY	RESET
AP_DEALLOC_ABEND	RESET
AP_DEALLOC_ABEND_PROG	RESET
AP_DEALLOC_ABEND_SVC	RESET
AP_DEALLOC_ABEND_TIMER	RESET
AP_DEALLOC_NORMAL	RESET
AP_PROG_ERROR_PURGING	RECEIVE
AP_SVC_ERROR_PURGING	RECEIVE

If the conversation is in RECEIVE state when the TP issues **SEND_ERROR**, incoming data is purged by APPC. This data includes:

- Data sent by SEND_DATA.
- Return code indicators.
- Confirmation requests.
- Deallocation requests.

APPC does not purge an incoming request-to-send indicator. APPC replaces purged incoming return code indicators with other return codes. The primary return code AP_OK replaces the following purged return code indicators:

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING

AP_PROG_ERROR_TRUNC

AP_SVC_ERROR_NO_TRUNC

AP_SVC_ERROR_PURGING

AP_SVC_ERROR_TRUNC

The primary return code AP_DEALLOC_NORMAL replaces the following purged return code indicators:

AP_ALLOCATION_ERROR

AP_ALLOCATION_FAILURE_NO_RETRY

AP_ALLOCATION_FAILURE_RETRY

AP_CONVERSATION_TYPE_MISMATCH

AP_DEALLOC_ABEND

AP_DEALLOC_ABEND_PROG

AP_DEALLOC_ABEND_SVC

AP_DEALLOC_ABEND_TIMER

AP_PIP_NOT_ALLOWED

AP_PIP_NOT_SPECIFIED_CORRECTLY

AP_SECURITY_NOT_VALID

AP_SYNC_LEVEL_NOT_SUPPORTED

AP_TP_NAME_NOT_RECOGNIZED

AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

AP_TRANS_PGM_NOT_AVAIL_RETRY

When the conversation is in SEND_PENDING state, APPC reports the following return codes to the partner TP based on the value in **err_dir**:

AP_PROG_ERROR_PURGING

The local TP issued **SEND_ERROR** with RECEIVE as the **err_dir**.

AP_PROG_ERROR_NO_TRUNC

The local TP issued **SEND_ERROR** with SEND as the **err_dir**.

AP_SVC_ERROR_PURGING

The local TP issued **SEND_ERROR** with RECEIVE as the **err_dir.**

AP_SVC_ERROR_NO_TRUNC

The local TP issued **SEND_ERROR** with SEND as the **err_dir**.

TEST_RTS

The **TEST_RTS** verb determines whether a request-to-send notification has been received from the partner transaction program (TP).

The following structure describes the verb control block (VCB) used by the **TEST_RTS** verb.

Syntax

```
struct test_rts {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       reserv3;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_TEST_RTS.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter was returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

Return Codes

AP_OK

Primary return code; the verb executed successfully.

AP_UNSUCCESSFUL

Primary return code; request-to-send notification has not been received.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **ALLOCATE**, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with **TP_STARTED** is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Remarks

The conversation can be in any state except RESET when the TP issues this verb.

There is no state change.

TEST_RTS_AND_POST

The **TEST_RTS_AND_POST** verb allows an application, typically a 5250 emulator, to request asynchronous notification when a partner transaction program (TP) requests send direction.

The following structure describes the verb control block (VCB) used by the **TEST_RTS_AND_POST** verb.

Syntax

```
struct test_rts {
    unsigned short      opcode;
    unsigned char       opext;
    unsigned char       reserv2;
    unsigned short      primary_rc;
    unsigned long       secondary_rc;
    unsigned char       tp_id[8];
    unsigned long       conv_id;
    unsigned char       reserv3;
    unsigned long       handle;
};
```

Members

opcode

Supplied parameter. Specifies the verb operation code, AP_B_TEST_RTS_AND_POST.

opext

Supplied parameter. Specifies the verb operation extension, AP_BASIC_CONVERSATION.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

tp_id

Supplied parameter. Identifies the local TP. The value of this parameter was returned by TP_STARTED in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

conv_id

Supplied parameter. Provides the conversation identifier. The value of this parameter was returned by ALLOCATE in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked TP.

reserv3

A reserved field.

handle

Supplied parameter. On Microsoft® Windows® 2000, this field provides the event handle to set.

Return Codes from Initial Verb

AP_OK

Primary return code; the verb executed successfully. Note particularly that a return code of AP_OK from the initial verb does not indicate that **REQUEST_TO_SEND** verb received from the partner TP. It simply indicates that the facility to receive asynchronous notification has been registered.

AP_UNSUCCESSFUL

Primary return code; request-to-send notification has not been received.

AP_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

AP_BAD_CONV_ID

Secondary return code; the value of **conv_id** did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

Secondary return code; the value of **tp_id** did not match a TP identifier assigned by APPC.

AP_INVALID_SEMAPHORE_HANDLE

Secondary return code, the value of **handle** was invalid.

AP_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

AP_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

When this return code is used with **ALLOCATE**, it may indicate that no communications system could be found to support the local logical unit (LU). (For example, the local LU alias specified with **TP_STARTED** is incorrect or has not been configured.) Note that if **lu_alias** or **mode_name** is fewer than eight characters, you must ensure that these fields are filled with spaces to the right. This error is returned if these parameters are not filled with spaces, since there is no node available that can satisfy the **ALLOCATE** request.

When **ALLOCATE** produces this return code for a Host Integration Server 2009 Client system configured with multiple nodes, there are two secondary return codes as follows:

0xF0000001

Secondary return code; no nodes have been started.

0xF0000002

Secondary return code; at least one node has been started, but the local LU (when **TP_STARTED** is issued) is not configured on any active nodes. The problem could be either of the following:

- The node with the local LU is not started.
- The local LU is not configured.

AP_CONVERSATION_TYPE_MIXED

Primary return code; the TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

AP_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

AP_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

AP_CONV_BUSY

Primary return code; there can only be one outstanding conversation verb at a time on any conversation. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **conv_id**.

AP_THREAD_BLOCKING

Primary return code; the calling thread is already in a blocking call.

AP_UNEXPECTED_DOS_ERROR

Primary return code; the operating system has returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

Return Codes from Asynchronous Completion
AP_OK

Primary return code; the request-to-send notification has been received from the partner TP.

AP_CANCELLED

The outstanding **TEST_RTS_AND_POST** verb has been terminated. This will occur if the underlying conversation has been deallocated or an AP_TP_ENDED has been issued. Note that as with **RECEIVE_AND_POST**, the TP is still responsible for correctly terminating the conversation and possibly terminating the TP. Issuing another verb, such as **RECEIVE_IMMEDIATE,** at this point will indicate the reason for the conversation failure.

The conversation can be in any state except RESET when the TP issues this verb. There is no state change.

A common feature of many APPC applications, such as 5250 emulators, is a requirement to detect a partner's request to send. Currently, this can be done by polling the APPC interface to detect the partner's request. For example, an application can occasionally issue one of the following verbs:

- **TEST_RTS**

Remarks

- **RECEIVE_IMMEDIATE** and check the **rts_rcvd** field
- **SEND_DATA** of zero bytes, again checking the **rts_rcvd** field.

Some of the problems associated with this polling approach are:

- The application must continually interrupt its main work to poll APPC.
- The partner's request is not detected as soon as it becomes available.
- These approaches are processor-intensive.

The **TEST_RTS_AND_POST** verb allows an application running on Windows 2000, typically a 5250 emulator, to request asynchronous notification when the partner TP requests send direction.

An APPC application typically issues the **TEST_RTS_AND_POST** verb while in SEND state and then continues with its main processing. A request for send direction from the partner TP is indicated asynchronously to the application. After dealing with the partner's request, the application typically returns to SEND state, reissues **TEST_RTS_AND_POST**, and continues.

The **TEST_RTS_AND_POST** verb completes synchronously and the return code AP_OK indicates that a request for asynchronous notification has been registered. It is important to emphasize that this does not indicate that request-to-send was received from the partner TP.

When the partner's request to send is received, the asynchronous event completion occurs. It is important to note that this may be before the completion of the local TP's original **TEST_RTS_AND_POST** verb. This will be the case if the partner's request to send was received before the local TP's **TEST_RTS_AND_POST** verb was issued, or while the local TP's **TEST_RTS_AND_POST** verb was being processed.

APPC Extensions for the Windows Environment

This section describes API extensions to Windows Advanced Program-to-Program Communications (APPC) that allow asynchronous communication. Asynchronous communication occurs when a function returns before the request completes. The application is notified later when the request is completed.

Under Microsoft® Windows® 2000, three methods are available for asynchronous communication using the APPC API:

- Message posting using window handles.
- Waiting on Win32® events.
- Using Win32 I/O completion ports.

The first method uses messages posted to a window handle to notify an application of verb completion. This method using window handles and messages was supported on Microsoft Windows 3.x. There is one such window for each APPC application, independent of the number of conversations. Each APPC conversation can have one asynchronous verb outstanding at any time. When a verb completes, the posting to the window takes as parameters the asynchronous task handle returned by the original call and a pointer to the verb control block which has completed, containing the return codes of the verb.

The extensions using window handles and message posting described in this section (WinAsyncAPPC) were designed for all implementations and versions of Microsoft Windows from version 3.0 through the latest versions of Windows 2000. They provided compatibility for Windows programming and optimum application performance in the 16-bit Windows operating environment.

A second method using Win32 events for notification is supported. The extensions using Win32 events described in this section (WinAsyncAPPCEx) operate only on Windows 2000, and offer optimum application performance in the 32-bit Windows operating environment. If an event has been registered with the conversation, then an application can call the Win32 **WaitForSingleObject** or **WaitForMultipleObjects** function to wait to be notified of the completion of the verb.

A third method using Win32 I/O completion ports for notification is supported on Windows 2000. The extensions using I/O completion ports described in this section (WinAsyncAPPCIOCP) operate only on Windows 2000, and offer optimum application performance in the 32-bit Windows operating environment. If an I/O completion port has been created using **CreateIoCompletionPort**, then an application can call the Win32 **GetQueuedCompletionStatus** function to wait to be notified of the completion of the verb.

Windows APPC allows multithreaded Windows-based processes. A process contains one or more threads of execution. All references to threads in this document refer to actual threads in multithreaded Windows environments.

This section provides, for each extension, a definition of the function, syntax, returns, and remarks for using the function.

In This Section

- WinAPPCUnhookBlockingHook

WinAsyncAPPC

The **WinAsyncAPPC** function provides an asynchronous entry point for all of the APPC verbs. Use this function instead of the blocking versions of the verbs if you run your application and want to use message posting using Windows handles for asynchronous verb completion.

Syntax

```
HANDLE WINAPI WinAsyncAPPC(
HANDLE hWnd,
Long lpVcb
);
```

Parameters

hWnd

A window handle that will be used for message posting to notify an application when an APPC verb completes.

lpVcb

Pointer to the verb control block.

Return Value

The return value specifies whether the asynchronous request was successful. If the function was successful, the return value is an asynchronous task handle. If the function was not successful, a zero is returned.

When this function returns with a successful value, this does not indicate that the APPC call will ultimately return successfully. It only indicates that it was possible for the APPC library to attempt the APPC call asynchronously using message posting for notification.

Remarks

For an example of how to use this verb in transaction programs (TPs), see the send and receive sample TP (SENDRECV.C located in the APPC folder) included in the SDK.

APPC verbs used in basic conversations that can block are as follows:

- ALLOCATE
- CONFIRM
- CONFIRMED
- DEALLOCATE
- FLUSH
- PREPARE_TO_RECEIVE
- RECEIVE_ALLOCATE
- RECEIVE_AND_WAIT
- REQUEST_TO_SEND
- SEND_CONVERSATION
- SEND_DATA
- SEND_ERROR

- TP_ENDED
- TP_STARTED

APPC verbs used in mapped conversations that can block are as follows:

- MC_ALLOCATE
- MC_CONFIRM
- MC_CONFIRMED
- MC_DEALLOCATE
- MC_FLUSH
- MC_PREPARE_TO_RECEIVE
- MC_RECEIVE_AND_WAIT
- MC_REQUEST_TO_SEND
- MC_SEND_CONVERSATION
- MC_SEND_DATA
- MC_SEND_ERROR

When using the synchronous or asynchronous versions of a verb, an application can only have one outstanding function in progress on a conversation at a time. An attempt to initiate a second function results in the error code AP_CONV_BUSY.

The exceptions to the preceding paragraph are:

- RECEIVE_AND_POST
- MC_RECEIVE_AND_POST
- RECEIVE_AND_WAIT
- MC_RECEIVE_AND_WAIT

To allow full use of the asynchronous support, asynchronously issued **RECEIVE_AND_WAIT** and **MC_RECEIVE_AND_WAIT** verbs have been altered to act like the **RECEIVE_AND_POST** and **MC_RECEIVE_AND_POST** verbs. Specifically, while an asynchronous version of one of these verbs is outstanding, the following verbs can be issued on the same conversation:

- DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
- GET_ATTRIBUTES or MC_GET_ATTRIBUTES
- GET_TYPE
- REQUEST_TO_SEND or MC_REQUEST_TO_SEND
- SEND_ERROR or MC_SEND_ERROR

- TEST_RTS or MC_TEST_RTS
- TP_ENDED

This allows an application, in particular, a 5250 emulator, to use an asynchronous **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_POST**, **MC_RECEIVE_AND_POST**, **RECEIVE_AND_WAIT**, or **MC_RECEIVE_AND_WAIT** is outstanding, it can still use **SEND_ERROR** or **MC_SEND_ERROR** and **REQUEST_TO_SEND** or **MC_REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support.

When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinAsyncAPPC" as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call. The *lParam* argument contains the original VCB pointer and can be dereferenced to determine the final return code.

As part of the Windows APPC definition, WinAPPCCancelAsyncRequest allows an application to cancel any asynchronous APPC action; but terminates the related conversation or TP as appropriate. Any outstanding operations return with AP_CANCELED as the return code.

If the function returns successfully, a **WinAsyncAPPC** message is posted to the application when the operation completes or the conversation is canceled.

WinAsyncAPPCEx

The **WinAsyncAPPCEx** function provides an asynchronous entry point for all of the APPC verbs. Use this function instead of the blocking versions of the verbs to allow multiple sessions to be handled on the same thread using events. This verb is only supported on Microsoft® Windows® 2000 and uses Win32® events.

Syntax

```
    HANDLE WINAPI WinAsyncAPPCEx(
HANDLEevent_handle,
longlpVcb);
```

Parameters

event_handle

Handle used for event notification using Win32 events.

lpVcb

Pointer to the verb control block.

Return Value

The return value specifies whether the asynchronous resolution request was successful. If the function was successful, the return value is an asynchronous task handle. If the function was not successful, a zero is returned.

When this function returns with a successful value, this does not indicate that the APPC call will ultimately return successfully. It only indicates that it was possible for the APPC library to attempt the APPC call asynchronously using events for notification.

Remarks

This function is intended for use with **WaitForSingleObject** or **WaitForMultipleObjects** in the Win32 API. These functions are described in the "Reference" section of the Microsoft Platform SDK documentation.

For an example of how to use this verb in multithreaded TPs, see the multithreaded send and receive sample TPs (MRCV.C, MSEND.C, and MSENDRCV.C located in the MSENDRCV folder) included in the SDK.

APPC verbs used in basic conversations that can block are as follows:

- ALLOCATE
- CONFIRM
- CONFIRMED
- DEALLOCATE
- FLUSH
- PREPARE_TO_RECEIVE
- RECEIVE_ALLOCATE
- RECEIVE_AND_WAIT
- REQUEST_TO_SEND
- SEND_CONVERSATION
- SEND_DATA

- SEND_ERROR
- TP_ENDED
- TP_STARTED

APPC verbs used in mapped conversations that can block are as follows:

- MC_ALLOCATE
- MC_CONFIRM
- MC_CONFIRMED
- MC_DEALLOCATE
- MC_FLUSH
- MC_PREPARE_TO_RECEIVE
- MC_RECEIVE_AND_WAIT
- MC_REQUEST_TO_SEND
- MC_SEND_CONVERSATION
- MC_SEND_DATA
- MC_SEND_ERROR
- RECEIVE_ALLOCATE
- TP_ENDED
- TP_STARTED

When using the synchronous or asynchronous versions of a verb, an application can only have one outstanding function in progress on a conversation at a time. An attempt to initiate a second function results in the error code AP_CONV_BUSY.

Note
The exceptions to the preceding paragraph are RECEIVE_AND_POST, MC_RECEIVE_AND_POST, RECEIVE_AND_WAIT, and MC_RECEIVE_AND_WAIT.

Note
To allow full use of the asynchronous support, asynchronously issued **RECEIVE_AND_WAIT** and **MC_RECEIVE_AND_WAIT** verbs have been altered to act like the **RECEIVE_AND_POST** and **MC_RECEIVE_AND_POST** verbs. Specifically, while an asynchronous version of one of these verbs is outstanding, the following verbs can be issued on the same conversation:

- DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
- GET_ATTRIBUTES or MC_GET_ATTRIBUTES
- GET_TYPE

- REQUEST_TO_SEND or MC_REQUEST_TO_SEND
- SEND_ERROR or MC_SEND_ERROR
- TEST_RTS or MC_TEST_RTS
- TP_ENDED

Note
This allows an application, in particular, a server application, to use an asynchronous **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_POST**, **MC_RECEIVE_AND_POST**, **RECEIVE_AND_WAIT**, or **MC_RECEIVE_AND_WAIT** is outstanding, it can still use **SEND_ERROR** or **MC_SEND_ERROR** and **REQUEST_TO_SEND** or **MC_REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support, and in particular, for support of multiple conversations on the same thread.

When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the APPC primary return code and secondary return code in the verb control block for any error conditions.

WinAsyncAPPCIOCP

The **WinAsyncAPPCIOCP** function provides an asynchronous entry point for all of the APPC verbs. Use this function instead of the blocking versions of the verbs to allow multiple sessions to be handled on the same thread using I/O completion ports. This verb is only supported on Microsoft Windows 2000, and uses Win32 I/O completion ports.

Syntax

```
    HANDLE WINAPI WinAsyncAPPCIOCP(
APPC_IOCP_INFO *iocp_handle,
longlpVcb);
```

Parameters

iocp_handle

A pointer to an **APPC_IOCP_INFO** structure used for passing I/O completion port information.

lpVcb

Pointer to the verb control block

The **APPC_IOCP_INFO** structure has the following prototype:

```
APPC_CompletionPort;APPC_NumberOfBytesTransferred;
    APPC_CompletionKey;
    APPC_pOverlapped;
```

APPC_CompletionPort

This supplied parameter is the HANDLE returned by the call to the **CreateIoCompletionPort** function when the I/O completion port is created. The I/O completion port must be created before calling the **WinAsyncAPPCIOCP** function. When the verb completes, the APPC Library calls the **PostQueuedCompletionStatus** function with the remaining fields in the structure as inputs, and these fields are simply passed through to the **GetQueuedCompletionStatus** function issued by the application.

APPC_NumberOfBytesTransferred

This supplied parameter is ignored. When the APPC verb completes, the APPC Library calls the **PostQueuedCompletionStatus** function with this field as an input, and the value returned for the *dwNumberOfBytesTransferred* is simply passed through to the **GetQueuedCompletionStatus** function issued by the application.

APPC_CompletionKey

This supplied parameter is ignored. When the APPC verb completes, the APPC Library calls the **PostQueuedCompletionStatus** function with this field as an input, and the value returned for the *dwCompletionKey* is simply passed through to the **GetQueuedCompletionStatus** function issued by the application.

APPC_pOverlapped

This supplied parameter is ignored. When the APPC verb completes, the APPC Library calls the **PostQueuedCompletionStatus** function with this field as an input, and the value returned for the *lpOverlapped* is simply passed through to the **GetQueuedCompletionStatus** function issued by the application.

Return Value

The return value specifies whether the asynchronous resolution request was successful. If the function was successful, the return value is an asynchronous task handle. If the function was not successful, a zero is returned.

When this function returns with a successful value, this does not indicate that the APPC call will ultimately return successfully. It only indicates that it was possible for the APPC library to attempt the APPC call asynchronously using an I/O completion port for notification.

Remarks

This function is intended for use with **CreateIoCompletionPort** and **GetQueuedCompletionStatus** in the Win32 API. These functions are described in the "Reference" section of the Microsoft Platform SDK documentation.

For an example of how to use this verb in multithreaded TPs, see the multithreaded receive sample TP (MRCVIO located in the SNA\MSENDRCV folder) using I/O completion ports included in the Host Integration Server 2009 SDK.

APPC verbs used in basic conversations that can block are as follows:

- ALLOCATE
- CONFIRM
- CONFIRMED
- DEALLOCATE
- FLUSH
- PREPARE_TO_RECEIVE
- RECEIVE_ALLOCATE
- RECEIVE_AND_WAIT
- REQUEST_TO_SEND
- SEND_CONVERSATION
- SEND_DATA
- SEND_ERROR
- TP_ENDED
- TP_STARTED

APPC verbs used in mapped conversations that can block are as follows:

- MC_ALLOCATE
- MC_CONFIRM
- MC_CONFIRMED
- MC_DEALLOCATE
- MC_FLUSH
- MC_PREPARE_TO_RECEIVE
- MC_RECEIVE_AND_WAIT
- MC_REQUEST_TO_SEND
- MC_SEND_CONVERSATION

- MC_SEND_DATA
- MC_SEND_ERROR
- RECEIVE_ALLOCATE
- TP_ENDED
- TP_STARTED

When using the synchronous or asynchronous versions of a verb, an application can only have one outstanding function in progress on a conversation at a time. An attempt to initiate a second function results in the error code AP_CONV_BUSY.

The exceptions to the preceding paragraph are RECEIVE_AND_POST, MC_RECEIVE_AND_POST, RECEIVE_AND_WAIT, and MC_RECEIVE_AND_WAIT.

To allow full use of the asynchronous support, asynchronously issued **RECEIVE_AND_WAIT** and **MC_RECEIVE_AND_WAIT** verbs have been altered to act like the **RECEIVE_AND_POST** and **MC_RECEIVE_AND_POST** verbs. Specifically, while an asynchronous version of one of these verbs is outstanding, the following verbs can be issued on the same conversation:

- DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
- GET_ATTRIBUTES or MC_GET_ATTRIBUTES
- GET_TYPE
- REQUEST_TO_SEND or MC_REQUEST_TO_SEND
- SEND_ERROR or MC_SEND_ERROR
- TEST_RTS or MC_TEST_RTS
- TP_ENDED

This allows an application, in particular, a server application, to use an asynchronous **RECEIVE_AND_WAIT** or **MC_RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_POST**, **MC_RECEIVE_AND_POST**, **RECEIVE_AND_WAIT**, or **MC_RECEIVE_AND_WAIT** is outstanding, it can still use **SEND_ERROR** or **MC_SEND_ERROR** and **REQUEST_TO_SEND** or **MC_REQUEST_TO_SEND**. It is recommended that you use this feature for full asynchronous support, and in particular, for support of multiple conversations on the same thread.

When the asynchronous operation is complete, the application is notified through the **GetQueuedCompletionStatus** function. Upon I/O completion, examine the APPC primary return code and secondary return code in the verb control block for any error conditions.

WinAPPCCancelAsyncRequest

The **WinAPPCCancelAsyncRequest** function cancels an outstanding WinAsyncAPPC-based request.

Syntax

```
    int WINAPI WinAPPCCancelAsyncRequest(
HANDLE hAsyncTaskID);
```

Parameters

hAsyncTaskID

Supplied parameter. Specifies the asynchronous task to be canceled.

Return Value

The return value specifies whether the asynchronous request was canceled. If the value is zero, the request was canceled. Otherwise, the value is one of the following:

WAPPCINVALID

An error code indicating that the specified asynchronous task identifier was invalid.

WAPPCALREADY

An error code indicating that the asynchronous routine being canceled has already completed.

Remarks

An asynchronous task previously initiated by issuing one of the **WinAsyncAPPC**, **WinAsyncAPPCEx**, or **WinAsyncAPPCIOCP** functions can be canceled prior to completion by issuing the **WinAPPCCancelAsyncRequest** function, specifying the asynchronous task identifier as returned by the initial function in *hAsyncTaskID*.

If the outstanding verb relates to a conversation (for example, SEND_DATA or RECEIVE_AND_WAIT), the verb is purged and the session is closed. If the verb relates to a TP (for example, RECEIVE_ALLOCATE or TP_STARTED), the TP is ended. In both cases, while the implementation closes conversations and sessions as cleanly as possible, it does not flush send buffers, wait for confirmations, and so on. This call is synchronous, and after the processing described above is complete, a completion message is posted for the canceled verb.

If an attempt to cancel an existing asynchronous **WinAsyncAPPC** routine fails with an error code of WAPPCALREADY, one of two things has occurred. Either the original routine has already completed and the application has dealt with the resulting message, or the original routine has already completed and the resulting message is still waiting in the application window queue.

WinAPPCCancelBlockingCall

The **WinAPPCCancelBlockingCall** function cancels any outstanding blocking operation for its thread. Any outstanding blocked call canceled will cause an error code of WAPPCCANCEL to be generated.

Syntax

```
BOOL WINAPI WinAPPCCancelBlockingCall(
void
);
```

Return Value

The return value specifies whether the cancellation request was successful. If the value is zero, the request was canceled. Otherwise, the value is the following:

WAPPCINVALID

An error code indicating that there is no outstanding blocking call.

Remarks

If the outstanding verb relates to a conversation (for example, SEND_DATA or RECEIVE_AND_WAIT), the verb is purged and the session is closed. If the verb relates to a TP (for example, RECEIVE_ALLOCATE or TP_STARTED), the TP is ended. In both cases, while the implementation brings down conversations and sessions as cleanly as possible, it does not flush send buffers, wait for confirmations, and so on. This call is synchronous and after the processing described above is complete, the function is finished.

In Microsoft® Windows® 2000, a multithreaded application can have multiple blocking operations outstanding, but only one per thread. To distinguish between multiple outstanding calls, **WinAPPCCancelBlockingCall** cancels the outstanding operation on the current, or calling, application thread if one exists; otherwise, it fails. By default in Windows 2000, Windows APPC suspends the calling application thread while an operation is outstanding. As a result, the thread on which the blocking operation was initiated will not regain control (and therefore, will not be able to issue a call to **WinAPPCCancelBlockingCall**) unless a blocking hook is registered for the thread using WinAPPCSetBlockingHook.

WinAPPCCleanup

The **WinAPPCCleanup** function terminates and deregisters an application from a Windows APPC implementation.

Syntax

```
BOOL WINAPI WinAPPCCleanup(
void
);
```

Return Value

The return value specifies whether the deregistration was successful. If the value is nonzero, the application was successfully deregistered. The application was not deregistered if a value of zero is returned.

Remarks

Use **WinAPPCCleanup** to indicate deregistration of a Windows APPC application from a Windows APPC implementation.

Conversations that are still active will be terminated and TPs ended. This function is equivalent to issuing TP_ENDED (HARD) on all TPs owned by the application.

See Also

Reference

WinAPPCStartup

WinAPPCIsBlocking

The **WinAPPCIsBlocking** function determines if a thread is executing while waiting for a previous blocking call to finish.

Syntax

```
BOOL WINAPI WinAPPCIsBlocking(
void
);
```

Return Value

The return value specifies the outcome of the function. If the value is nonzero, there is an outstanding blocking call awaiting completion. A zero indicates the absence of an outstanding blocking call.

Remarks

Although a call issued on a blocking function appears to an application as though it blocks, the Windows APPC DLL has to relinquish the processor to allow other applications to run. This means that it is possible for the application that issued the blocking call to be re-entered, depending on the message(s) it receives. In this instance, the **WinAPPCIsBlocking** call can be used to determine whether the application task currently has been re-entered while waiting for an outstanding blocking call to finish. Note that Windows APPC prohibits more than one outstanding blocking call per thread.

The Windows APPC DLL prohibits more than one blocking call per thread and returns AP_THREAD_BLOCKING if this occurs.

See Also

Reference

WinAPPCSetBlockingHook
WinAPPCUnhookBlockingHook
WinAPPCCancelBlockingCall

WinAPPCStartup

The **WinAPPCStartup** function allows an application to specify the version of Windows APPC required and to retrieve details of the specific Windows APPC implementation. An application must call this function to register itself with a Windows APPC implementation before issuing any further Windows APPC calls.

Syntax

```
            int WINAPI WinAPPCStartup(
        WORDwVersionRequired,
        LPWAPPCDATAlpAPPCData
);

typedef struct {
    WORD wVersion;
    char szDescription[WAPPCDESCRIPTION_LEN+1];
} WAPPCDATA, FAR * LPWAPPCDATA;

where WAPPCDESCRIPTION_LEN is defined as 127
```

Parameters

wVersionRequired

Specifies the version of Windows APPC support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number. The current version of the Windows APPC API is 1.0.

lpAPPCData

Pointer to a returned structure containing a Windows APPC version number and a description of the Windows APPC implementation.

Return Value

The return value specifies whether the application was registered successfully and whether the Windows APPC implementation can support the specified version number. If the value is zero, it was registered successfully and the specified version can be supported. Otherwise, the return value is one of the following:

WAPPCSYSNOTREADY

The underlying network system is not ready for network communication.

WAPPCVERNOTSUPPORTED

The version of Windows APPC support requested is not provided by this particular Windows APPC implementation.

WAPPCINVALID

The Windows APPC version specified by the application is not supported by this DLL.

Remarks

To support future Windows APPC implementations and applications that may have functionality differences from Windows APPC version 1.0, a negotiation takes place in **WinAPPCStartup**. An application passes to **WinAPPCStartup** the Windows APPC version that it can use. If this version is lower than the lowest version supported by the Windows APPC DLL, the DLL cannot support the application and **WinAPPCStartup** fails. If the version is not lower, however, the call succeeds and returns the highest version of Windows APPC supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows APPC DLL on the system.

This negotiation allows both a Windows APPC DLL and a Windows APPC application to support a range of Windows APPC versions. An application can successfully use a DLL if there is any overlap in the versions. The following table illustrates how **WinAPPCStartup** works in conjunction with different application and DLL versions.

Application versions	DLL versions	To WinAPPCStartup	From WinAPPCStartup	Result
1.0	1.0	1.0	1.0	Use 1.0

1.0, 2.0	1.0	2.0	1.0	Use 1.0
1.0	1.0, 2.0	1.0	2.0	Use 1.0
1.0	2.0, 3.0	1.0	WAPPCINVALID	Fail
2.0, 3.0	1.0	3.0	1.0	App Fails
1.0, 2.0, 3.0	1.0, 2.0, 3.0	3.0	3.0	Use 3.0

Details of the actual Windows APPC implementation are described in the **WAPPCDATA** structure defined as follows that is returned by **WinAPPCStartup**:

```
typedef struct tagWAPPCDDATA { WORD wVersion;
char szDescription[WAPPCDESCRIPTION_LEN+1];
} WAPPCDATA, FAR *LPWAPPCDATA;
```

The structure members are as follows:

wVersion

The highest APPC version number supported by the Windows APPC DLL.

szDescription

A descriptive string describing the WinAPPC implementation.

After it makes its last Windows APPC call, an application should call the WinAPPCCleanup routine.

Each Windows APPC implementation must make a **WinAPPCStartup** call before issuing any other Windows APPC calls.

WinAPPCSetBlockingHook

The **WinAPPCSetBlockingHook** function allows a Windows APPC implementation to block APPC function calls by means of a new function. By default in Microsoft Windows 2000, blocking calls suspend the calling application's thread until the request is finished.

Syntax

```
        FARPROC WINAPI WinAPPCSetBlockingHook (
    FARPROC lpBlockFunc);
```

Parameters

lpBlockFunc

Specifies the procedure instance address of the blocking function to be installed.

Return Value

The return value points to the procedure instance of the previously installed blocking function. The application or library that calls **WinAPPCSetBlockingHook** should save this return value so that it can be restored if needed. (If nesting is not important, the application can simply discard the value returned by **WinAPPCSetBlockingHook** and eventually use WinAPPCUnhookBlockingHook to restore the default mechanism.)

Remarks

A Windows APPC implementation has a default mechanism by which blocking APPC functions are implemented. This function gives the application the ability to execute its own function at blocking time in place of the default function.

The default blocking function is equivalent to:

```
BOOL DefaultBlockingHook (void)  {
    MSG msg;
    /* get the next message if any */
    if ( PeekMessage (&msg,0,0,PM_NOREMOVE)  )  {
        if ( msg.message = = WM_QUIT  )
            return FALSE;   // let app process WM_QUIT
        PeekMessage (&msg,0,0,PM_REMOVE) ;
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    /* TRUE if no WM_QUIT received */
    return TRUE;
}
```

A blocking function must return FALSE if it receives a WM_QUIT message so Windows APPC can return control to the application to process the message and terminate gracefully. Otherwise, the function should return TRUE.

This function is implemented on a per-thread basis. It provides for a particular thread to replace the blocking mechanism without affecting other threads.

The **WinAPPCSetBlockingHook** function is provided to support those applications that require more complex message processing—for example, those employing the multiple document interface (MDI) model.

See Also

Reference

WinAPPCIsBlocking

WinAPPCCancelBlockingCall

WinAPPCUnhookBlockingHook

The **WinAPPCUnhookBlockingHook** function removes any previous blocking hook that has been installed and reinstalls the default blocking mechanism.

Syntax

```
BOOL WINAPI WinAPPCUnhookBlockingHook(
void
);
```

Return Value

The return value specifies the outcome of the function. It is nonzero if the default mechanism is successfully reinstalled. The value is zero if the mechanism did not reinstall.

See Also

Reference

WinAPPCSetBlockingHook

Host Integration Server Enhancements to the Windows Environment

This section describes the extensions to Windows Advanced Program-to-Program Communications (APPC) and the Common Service Verb (CSV) API that are specific to Host Integration Server 2009.

The GetAppcConfig function takes a local logical unit (LU) and returns the remote LUs that are accessible to the user through that LU. If left blank, and a default local LU has been configured, the user's default local LU will be used. In all instances, if one of the returned remote LUs is the user's default, it is indicated as such.

The call is asynchronous and completion is normally signaled by the posting of a Microsoft Windows message. However, an alternative completion mechanism is provided for console applications.

The GetAppcReturnCode and GetCsvReturnCode functions convert the primary and secondary return codes in the verb control block (VCB) to a printable string. These functions provide a standard set of error strings for use by applications.

For each extension, this section provides a definition of the function, syntax, returns, and remarks for using the function.

In This Section

GetAppcConfig

The **GetAppcConfig** function provides an asynchronous entry point for retrieving the remote systems to which a particular local LU can connect.

Syntax

```
HANDLE WINAPI GetAppcConfig(
HANDLE hWnd,
LPSTR pLocalLu,
LPSTR pMode,
LPINT pNumRemLu,
INT iMaxRemLu,
PSTR pRemLu,
LPINT pAsyncRetCode
);
```

Parameters

hWnd

Supplied parameter. Contains the handle of the window that is to receive an asynchronous completion message when the call has completed. If non-null, the completion message will be posted to this window handle. In this case, *pAsyncRetCode* (the last parameter) must be null. Asynchronous message completion is the recommended approach for Windows applications to use this function.

pLocalLu

Supplied parameter. Specifies the address of a buffer containing the local LU name for which information is returned. This local LU name must be specified as follows:

- Nonpadded
- Null-terminated
- ASCII string
- Maximum length of eight bytes (excluding the terminator)

To request that the user's default local LU be used, the buffer should contain eight spaces followed by a null.

pMode

Supplied parameter. Specifies the address of a buffer containing the mode name for which information is returned. In Microsoft Host Integration Server, this parameter is not used, but for compatibility with earlier versions of SNA Server a mode name must be specified as follows:

- Nonpadded
- Null-terminated
- ASCII string
- Maximum length of eight bytes (excluding the terminator)

pNumRemLu

Supplied parameter. Specifies the address of an integer variable that when the function completes will contain the number of remote LUs that would have been returned, had the buffer specified by *pRemLu* been large enough to accommodate all of the remote LUs.

iMaxRemLu

Supplied parameter. Specifies the number of remote LU names that can be held by the buffer indicated by *pRemLu*.

pRemLu

Supplied parameter. Specifies the address of the buffer that will hold the remote LU names after the function completes. The information will be returned as an array of strings. Each remote LU name will be stored in the buffer as follows:

- Nonpadded
- Null-terminated
- ASCII string
- Maximum length of eight bytes (excluding the terminator)

The strings start every nine bytes in the buffer, and thus (*pRemLu* + (*i*–1)*9) gives the start of the *i*th string. In the case where the buffer is too small to hold all the names, only *iMaxRemLu* strings will be returned.

pAsyncRetCode

Supplied parameter. Specifies the address of an integer variable used to store the return code from this function, if the supplied address is non-null. The return codes will be the same as those returned by an asynchronous completion message. While the call is completing, the value of this variable will be APPC_CFG_PENDING. When this asynchronous call is completed, the value of this variable will contain some return code other than APPC_CFG_PENDING.

This variable is used by polling for completion when asynchronous message completion to a window handle is not used.

Note that if *pAsyncRetCode* is used, *hWnd* must be null.

Return Value

The meaning of the immediate return value depends on whether or not the asynchronous request was accepted. To test for acceptance, evaluate the expression:

(<*Returned Handle*> & APPC_CFG_SUCCESS)

If the expression is FALSE, the request was rejected. The return value is then one of the synchronous return codes in the following list. If the expression is TRUE, the request was accepted, and one of the following cases will apply.

- If *hWnd* was non-null, a completion message will arrive in the following form:

Message parameter	**Description**
hWnd	The handle of the target window. This value is the same as the value passed in *hWnd* on the initial call.
uMsg	Matches the number returned by a call to **RegisterWindowMessage**, with WinAppcCfg used as the identifying string. This string is available by the **#define WIN_APPC_CFG_COMPLETION_MSG**.
wParam	Matches the *HANDLE* returned from the initial call. It is used as a correlator.
lParam	Contains one of the asynchronous return codes in the following list.

- If *pAsyncRetCode* was non-null, then the specified integer variable will be set to APPC_CFG_PENDING. After this function completes asynchronously, its value will change to one of the asynchronous return codes listed below.

Synchronous Return Codes
APPC_CFG_ERROR_NO_APPC_INIT

The Windows APPC library needs to be initialized by a call to WinAPPCStartup before calling **GetAppcConfig** and this has not been done.

APPC_CFG_ERROR_INVALID_HWND

The handle passed in *hWnd* was non-null, yet not a valid window handle.

APPC_CFG_ERROR_BAD_POINTER

The *hWnd* parameter was null, indicating that completion was signaled by setting the integer variable pointed to by *pAsyncRetCode*, but *pAsyncRetCode* was not a valid pointer.

APPC_CFG_ERROR_UNCLEAR_COMPLETION_MODE

Both *hWnd* and *pAsyncCompletion* were non-null, so **GetAppcConfig** was unable to decide how completion should be signaled.

APPC_CFG_ERROR_TOO_MANY_REQUESTS

Too many **GetAppcConfig** calls are already being processed (currently, this indicates 16 requests are outstanding). Try the call again after a delay. For the Microsoft Windows version 3.*x* system, you must yield during this period.

APPC_CFG_ERROR_GENERAL_FAILURE

An unexpected error occurred, probably of a system nature.

Asynchronous Return Codes
APPC_CFG_SUCCESS_NO_DEFAULT_REMOTE

The configuration information has been retrieved, and either no default remote LU was defined or it was not accessible by the specified local LU.

APPC_CFG_SUCCESS_DEFAULT_REMOTE

The configuration information has been retrieved, and there is a default remote LU that is accessible by the specified local LU.

APPC_CFG_ERROR_NO_DEFAULT_LOCAL_LU

An attempt was made to retrieve remote LUs partnered with the default local LU, but no default local LU was configured.

APPC_CFG_ERROR_BAD_LOCAL_LU

The local LU specified is either not configured, or is not valid for the calling verb.

APPC_CFG_ERROR_GENERAL_FAILURE

An unexpected error occurred, probably of a system nature.

Remarks

WinAPPCStartup must be called before using **GetAppcConfig**.

Whether an error code represents success or failure can be determined by evaluating either (*RetCode*& APPC_CFG_SUCCESS) to test for success or (*RetCode*& APPC_CFG_FAILURE) to test for failure.

The following code fragment shows how a console application can test completion:

```
while (*pAsyncRetCode == APPC_CFG_PENDING)
{
    sleep(250);
}
```

GetAppcReturnCode

The **GetAppcReturnCode** function converts the primary and secondary return codes in the verb control block to a printable string. This function provides a standard set of error strings for use by APPC applications such as 5250 emulators.

Syntax

```
    int WINAPI GetAppcReturnCode(
struct appc_hdr FAR * vpb,
UINTbuffer_length,
unsigned char FAR * buffer_addr);
```

Parameters

vpb

Supplied parameter. Specifies the address of the verb control block.

buffer_length

Supplied parameter. Specifies the length of the buffer pointed to by *buffer_addr*. The recommended length is 256.

buffer_addr

Supplied parameter. Specifies the address of the buffer that will hold the formatted, null-terminated string.

Return Value

The **GetAppcReturnCode** function returns a positive value on success that indicates the length of the error string passed back in *buffer_addr*.

A return value of zero indicates an error. On Microsoft® Windows® 2000 a call to **GetLastError** provides the actual error return code as follows:

0x20000001

The parameters are invalid; the function could not read from the specified verb control block or could not write to the specified buffer.

0x20000002

The specified buffer is too small.

0x20000003

The APPC string library APPCST32.DLL could not be loaded.

Remarks

The descriptive error string returned in *buffer_addr* does not terminate with a new line character (**\n**).

The descriptive error strings are contained in APPCST32.DLL and can be customized for different languages.

GetCsvReturnCode

The **GetCsvReturnCode** function converts the primary and secondary return codes in the verb control block to a printable string. This function provides a standard set of error strings for use by applications using common service verbs (CSVs).

Syntax

```
    int WINAPI GetCsvReturnCode(
struct csv_hdr FAR * vpb,
UINTbuffer_length,
unsigned char FAR * buffer_addr);
```

Parameters

vpb

Supplied parameter. Specifies the address of the verb control block.

buffer_length

Supplied parameter. Specifies the length of the buffer pointed to by *buffer_addr*. The recommended length is 256.

buffer_addr

Supplied parameter. Specifies the address of the buffer that will hold the formatted, null-terminated string when the function completes.

Return Value

The **GetCsvReturnCode** function returns a positive value on success that indicates the length of the error string passed back in *buffer_addr*.

A return value of zero indicates an error. On Microsoft® Windows® 2000 a call to **GetLastError** provides the actual error return code as follows:

0x20000001

The parameters are invalid; the function could not read from the specified verb parameter block or could not write to the specified buffer.

0x20000002

The specified buffer is too small.

0x20000003

The CSV string library CSVST32.DLL could not be loaded.

Remarks

The descriptive error string returned in *buffer_addr* does not terminate with a newline character (**\n**).

The descriptive error strings are contained in CSVST32.DLL and can be customized for different languages.

Common Service Verbs

This section describes each of the common service verbs (CSVs) and provides:

- A definition of the verb.
- The structure that defines the verb control block (VCB) used by the verb. The structure is declared in the WINCSV.H file.
- The parameters (VCB fields) supplied to and returned by the verb. A description of each parameter is provided, along with its possible values and other information.
- Additional information describing the use of the verb.

Most parameters supplied to and returned by CSVs are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants, which are established by **#define** statements in the header file WINCSV.H. For example, the **opcode** (operation code) parameter for CONVERT is the hexadecimal value represented by the symbolic constant SV_CONVERT. Use only the symbolic constants when programming CSVs.

This section contains:

CONVERT

The **CONVERT** verb translates an ASCII character string to EBCDIC or an EBCDIC character string to ASCII. The string to be converted is called the source string. The converted string is called the target string.

The following structure describes the verb control block (VCB) used by the **CONVERT** verb.

Syntax

```
struct convert {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        direction;
    unsigned char        char_set;
    unsigned short       len;
    unsigned char FAR *  source;
    unsigned char FAR *  target;
};
```

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_CONVERT.

opext

A reserved field.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

direction

Supplied parameter. Specifies the direction of the conversion. To convert from ASCII to EBCDIC, use SV_ASCII_TO_EBCDIC. To convert from EBCDIC to ASCII, use SV_EBCDIC_TO_ASCII.

char_set

Supplied parameter. Specifies the character set to use in converting the source string. Allowed values include SV_A (type A character set), SV_AE (type AE character set), and SV_G (user-defined type G character set).

len

Supplied parameter. Specifies the number of characters to be converted.

This length plus the offset from the beginning of the source or target buffer must not exceed the segment boundary.

source

Supplied parameter. Specifies the address of the buffer containing the character string to be converted.

target

Supplied parameter. Specifies the address of the buffer to contain the converted character string.

This buffer can overlap or coincide with the buffer pointed to by the **source** parameter. In this case, the converted data string

overwrites the source data string.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_CONVERSION_ERROR

Secondary return code; one or more characters in the source string were not found in the conversion table. These characters were converted to nulls (0x00). The verb still executed.

SV_INVALID_CHARACTER_SET

Secondary return code; the **char_set** parameter contained an invalid value.

SV_INVALID_DATA_SEGMENT

Secondary return code; the data buffer containing the source or target string did not fit in one segment, or the target segment was not a read/write segment.

SV_INVALID_DIRECTION

Secondary return code; the direction contained an invalid value.

SV_INVALID_FIRST_CHARACTER

Secondary return code; the first character of a type A source string was invalid.

SV_TABLE_ERROR

Secondary return code; one of the following occurred:

- The file containing the user-written type G conversion table was not specified by the environment variable CSVTBLG.
- The table was not in the correct format.
- The file specified by the CSVTBLG variable was not found.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.
- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.
- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

Remarks

The type A character set consists of:

- Uppercase letters.
- Numerals 0 through 9.
- Special characters $, #, @, and space.

This character set is supported by a system-supplied type A conversion table.

The first character of the source string must be an uppercase letter or the special character $, #, or @. Spaces are allowed only in trailing positions. Lowercase ASCII letters are translated to uppercase EBCDIC letters when the direction is ASCII to EBCDIC.

The type AE character set consists of:

- Uppercase letters.
- Lowercase letters.
- Numerals 0 through 9.
- Special characters $, #, @, period, and space.

This character set is supported by a system-supplied type AE conversion table.

The first character of the source string can be any character in the character set, except the space. Spaces are allowed only in trailing positions.

During conversion, embedded blanks (including blanks in the first position) are converted to 0x00. Although such a conversion will complete, CONVERSION_ERROR is returned as the secondary return code, indicating that the CSV library has completed an irreversible conversion on the supplied data.

For Windows 2000 a description of COMTBLG should point to the Windows 2000 registry under **\SnaBase\Parameters\Client**.

The data for a type G conversion table must be an ASCII file 32 lines long. Each line must consist of 32 hexadecimal digits, representing 16 characters, and be terminated by a carriage return and line feed. The first 16 lines (256 characters) specify the EBCDIC characters to which ASCII characters are converted; the remaining 16 lines specify the ASCII characters to which EBCDIC characters are converted.

The hexadecimal digits A through F can be either uppercase or lowercase. However, you may want to make these digits uppercase to ensure compatibility with IBM ES for OS/2 version 1.0.

Note
You can use GET_CP_CONVERT_TABLE to build a type G user-written conversion table in memory, and then store the table in a file.

COPY_TRACE_TO_FILE

The **COPY_TRACE_TO_FILE** verb concatenates individual API/link service trace files to form a single file.

The following structure describes the verb control block (VCB) used by the **COPY_TRACE_TO_FILE** verb.

Syntax

```
struct copy_trace_to_file {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        reserv3[8];
    unsigned char        file_name[64];
    unsigned char        file_option;
    unsigned char        reserv4[12];
};
```

Remarks

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_COPY_TRACE_TO_FILE.

opext

A reserved field.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

reserv3

A reserved field.

file_name

Supplied parameter. Specifies the name of the file to which trace data is to be copied. This parameter is a 64-byte character string, and it can include a path. If the name is fewer than 64 bytes, use spaces to pad it on the right.

file_option

Supplied parameter. Specifies the output file copy option:

- Use SV_NEW to copy the trace only if the specified file does not already exist.
- Use SV_OVERWRITE to copy the trace to an existing file, overwriting the current data. The size of the file is increased if necessary; and the file is created if it does not already exist.

reserv4

The address at which supplied data resides.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_INVALID_FILE_OPTION

Secondary return code; a value other than SV_NEW or SV_OVERWRITE was specified for **file_option**.

SV_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

SV_COPY_TRACE_IN_PROGRESS

Secondary return code; a previously issued **COPY_TRACE_TO_FILE** verb is still in progress.

SV_TRACE_FILE_EMPTY

Secondary return code; there is no data in the trace files.

SV_TRACE_NOT_STOPPED

Secondary return code; a trace was in progress when the verb was issued.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_FILE_ALREADY_EXISTS

Primary return code; when the SV_NEW file option was used, the file name specified was the name of an existing file.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_OUTPUT_DEVICE_FULL

Primary return code; there is insufficient space on the device where the output file resides. Retry the operation after freeing additional disk space.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.

- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.

- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

Remarks

There are two API/link-service trace files. The files are used alternately; tracing switches from one file to the other when one file is full (larger than 250K). When **COPY_TRACE_TO_FILE** is called, these trace files are concatenated and copied to a single file, the name of which is specified as a parameter to the call.

API/link-service tracing is stopped before issuing the verb, and restarted after the copy is complete. The trace files are reset when this verb is successfully completed.

DEFINE_TRACE

The **DEFINE_TRACE** verb enables or disables tracing for specified APIs and controls the amount of tracing.

The following structure describes the verb control block (VCB) used by the **DEFINE_TRACE** verb.

Syntax

```
struct define_trace {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        reserv3[8];
    unsigned char        dt_set;
    unsigned char        appc;
    unsigned char        reserv4;
    unsigned char        srpi;
    unsigned char        sdlc;
    unsigned char        tkn_rng_dlc;
    unsigned char        pcnet_dlc;
    unsigned char        dft;
    unsigned char        acdi;
    unsigned char        reserv5;
    unsigned char        ehllapi;
    unsigned char        x25_api;
    unsigned char        x25_dlc;
    unsigned char        twinax;
    unsigned char        reserv6;
    unsigned char        lua_api;
    unsigned char        etherand;
    unsigned char        subsym;
    unsigned char        reserv7[8];
    unsigned char        reset_trc;
    unsigned short       trunc;
    unsigned short       strg_size;
    unsigned char        reserv8;
    unsigned char        phys_link[8];
    unsigned char        reserv9[56];
};
```

Remarks

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_DEFINE_TRACE.

opext

A reserved field.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

reserv3

A reserved field.

dt_set

Supplied parameter. Sets the trace state.

- Use SV_ON to enable tracing for a particular API if the parameter pertaining to the API (such as **appc** or **comm_serv**) is set to SV_CHANGE.
- Use SV_OFF to disable tracing for a particular API if the parameter pertaining to the API is set to SV_CHANGE.

appc

Supplied parameter. Indicates whether tracing of APPC is desired.

- Use SV_CHANGE to enable or disable tracing for APPC, depending on the **dt_set** parameter.
- Use SV_IGNORE to leave tracing in its current state for APPC.

The allowed values turn bit 0 on or off; bits 1 through 7 are reserved.

reserv4

A reserved field.

srpi

Supplied parameter. Indicates whether tracing of SRPI is desired.

- Use SV_CHANGE to enable or disable tracing for APPC, depending on the **dt_set** parameter.
- Use SV_IGNORE to leave tracing in its current state for APPC.

sdlc

A reserved field.

tkn_rng_dlc

A reserved field.

pcnet_dlc

A reserved field.

dft

A reserved field.

acdi

A reserved field.

reserv5

A reserved field.

comm_serv

Supplied parameter. Indicates whether tracing of COMM_SERV_API is desired.

- Use SV_CHANGE to enable or disable tracing for APPC, depending on the **dt_set** parameter.
- Use SV_IGNORE to leave tracing in its current state for APPC.

ehllapi

A reserved field.

x25_api

A reserved field.

x25_dlc

A reserved field.

twinax

A reserved field.

reserv6

A reserved field.

lua_api

A reserved field.

etherand

A reserved field.

subsym

A reserved field.

reserv7

A reserved field.

reset_trc

Supplied parameter. Indicates whether the trace file pointer should be reset.

- Use SV_NO to not reset the trace file pointer to the start of the trace file. Previous trace records are not overwritten.
- Use SV_YES to reset the trace file pointer to the start of the trace file. Previous trace records are overwritten.

trunc

Supplied parameter. Specifies the maximum number of bytes for each trace record. Excess bytes are truncated. Set this value to zero if you do not want truncation.

strg_size

A reserved field.

reserv8

A reserved field.

phys_link

A reserved field.

reserv9

A reserved field.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_INVALID_RESET_TRACE

Secondary return code; the **reset_trc** parameter contained an invalid value.

SV_INVALID_SET

Secondary return code; the **dt_set** parameter contained an invalid value.

SV_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

SV_COPY_TRACE_IN_PROGRESS

Secondary return code; a previously issued COPY_TRACE_TO_FILE is still in progress. Traces cannot be active while using **DEFINE_TRACE**.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.

- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.

- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

Remarks

For information on how to run and use traces, see the appropriate manual for your product.

GET_CP_CONVERT_TABLE

The **GET_CP_CONVERT_TABLE** verb creates and returns a 256-byte conversion table to translate character strings from a source code page to a target code page.

The following structure describes the verb control block (VCB) used by the **GET_CP_CONVERT_TABLE** verb.

Syntax

```
struct get_cp_convert_table {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        reserv3[8];
    unsigned short       source_cp;
    unsigned short       target_cp;
    unsigned char FAR *  conv_tbl_addr;
    unsigned char        char_not_fnd;
    unsigned char        substitute_char;
};
```

Remarks

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_GET_CP_CONVERT_TABLE.

opext

A reserved field.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

reserv3

A reserved field.

source_cp

Supplied parameter. Specifies the source code page from which characters are converted. The allowed code pages (decimal values) are as follows:

- ASCII 437, 850, 860, 863, 865
- EBCDIC 037, 273, 277, 278, 280, 284, 285, 297, 500

User-defined code pages in the range from 65280 through 65535 are also allowed.

ASCII code pages are sometimes referred to as PC code pages; EBCDIC code pages are sometimes referred to as host code pages.

target_cp

Supplied parameter. Specifies the target code page to which characters are converted. For allowed code pages, see the preceding definition for **source_cp**.

conv_tbl_addr

Supplied parameter. Specifies the address of the buffer to contain the 256-byte conversion table. The buffer must be in a writable segment and long enough to contain the table.

char_not_fnd

Supplied parameter. Specifies the action to take if a character in the source code page does not exist in the target code page:

- Use SV_ROUND_TRIP to store a unique value in the conversion table for each source code page character.
- Use SV_SUBSTITUTE to store a substitute character (specified by **substitute_char**) in the conversion table.

substitute_char

Supplied parameter. Specifies the character to store in the conversion table when a character from the source code page has no equivalent in the target code page.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_INVALID_CHAR_NOT_FOUND

Secondary return code; the **char_not_fnd** parameter contained an invalid value.

SV_INVALID_DATA_SEGMENT

Secondary return code; the 256-byte area specified for the conversion table extended beyond the segment boundary, or the segment was not writable.

SV_INVALID_SOURCE_CODE_PAGE

Secondary return code; the code page specified by **source_cp** is not supported.

SV_INVALID_TARGET_CODE_PAGE

Secondary return code; the code page specified by **target_cp** is not supported.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.
- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.
- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has

been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

Remarks

The type A character set consists of:

- Uppercase letters.
- Numerals 0 through 9.
- Special characters $, #, @, and space.

This character set is supported by a system-supplied type A conversion table.

The first character of the source string must be an uppercase letter or the special character $, #, or @. Spaces are allowed only in trailing positions. Lowercase ASCII letters are translated to uppercase EBCDIC letters when the direction is ASCII to EBCDIC.

The type AE character set consists of:

- Uppercase letters.
- Lowercase letters.
- Numerals 0 through 9.
- Special characters $, #, @, period, and space.

This character set is supported by a system-supplied type AE conversion table.

The first character of the source string can be any character in the character set except the space.

During conversion, embedded blanks (including blanks in the first position) are converted to 0x00. Although such a conversion will complete, CONVERSION_ERROR is returned as the secondary return code, indicating that the CSV library has completed an irreversible conversion on the supplied data.

For Windows 2000, a description of COMTBLG should point to the Windows 2000 registry under **\SnaBase\Parameters\Client**. For the OS/2 operating system, the directory and file containing the table must be specified by the environment variable COMTBLG. (If the file is not found, the system returns the SV_TABLE_ERROR parameter check.).

The SV_ROUND_TRIP value for **char_not_fnd** is useful only if you build a second conversion table to convert between the same two code pages in the reverse direction. If you specify the SV_ROUND_TRIP value in building both conversion tables, any character translated from one code page to the other and then back will be unchanged.

When using the SV_SUBSTITUTE value for **char_not_fnd**, converting the translated character string back to the original code page will not necessarily re-create the original character string.

Use **substitute_char** only if **char_not_fnd** is set to SV_SUBSTITUTE.

The value stored in the conversion table is the ASCII value associated with the character. If the table is used for conversion from ASCII to EBCDIC, the character that appears in the converted string is the character associated with the numeric EBCDIC value rather than ASCII.

For example, if you supply the underscore (_) character (ASCII value F6) while creating an ASCII to EBCDIC conversion table, the character that appears in the converted strings will be 6, the character associated with the value F6 in EBCDIC. To use the _ character as the substitute character in an ASCII to EBCDIC conversion table, you should supply the value E1 (the value associated with the _ character in EBCDIC) rather than the actual character.

A code page is a table that associates specific ASCII or EBCDIC values with specific characters. If a character from the source code page does not exist in the target code page, the translated (target) string differs from the original (source) string.

LOG_MESSAGE

For OS/2 only, the **LOG_MESSAGE** verb records a message in the error log file and optionally displays the message on the users screen. This verb is included for compatibility with existing applications.

The following structure describes the verb control block (VCB) used by the **LOG_MESSAGE** verb.

Syntax

```
struct log_message {
    unsigned short       opcode;
    unsigned char        opext;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned short       msg_num;
    unsigned char        orignтr_id[8];
    unsigned char        msg_file_name[3];
    unsigned char        msg_act;
    unsigned short       msg_ins_len;
    unsigned char FAR *  msg_ins_ptr;
};
```

Remarks

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_LOG_MESSAGE.

opext

A reserved field.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

msg_num

Supplied parameter. Specifies the number of the message in the message file specified by **msg_file_name**.

orignтr_id

Supplied parameter. Specifies the name of the component issuing **LOG_MESSAGE** or an 8-byte, user-supplied string.

msg_file_name

Supplied parameter. Specifies the name of the file containing the message to be logged.

msg_act

Supplied parameter. Specifies the action to be taken when processing the message:

- Use SV_INTRV to log the intervention with a severity level of 12 and display the message on the users screen. The user must press a key to remove the message from the screen.
- Use SV_NO_INTRV to log the intervention with a severity level of 12 but not display the message.

msg_ins_len

Supplied parameter. Specifies the length of data to be inserted into the message. Set this parameter to zero if no data is to be inserted.

msg_ins_ptr

Supplied parameter. Specifies the address of the data to be inserted into the message.

Use this parameter only if **msg_ins_len** is greater than zero.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_INVALID_DATA_SEGMENT

Secondary return code; the data that was to be inserted into the message extended beyond the segment boundary.

SV_INVALID_MESSAGE_ACTION

Secondary return code; the **msg_act** parameter contained an invalid value.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.
- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.
- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

Remarks

The value for **msg_file_name** must be three characters long. Pad with spaces if necessary. The .MSG extension is added automatically.

The total length of **msg_ins_len**, including header information (40 bytes), message text, and inserted data, should not exceed 256 bytes. If the length is greater than 256 bytes, the communication system will attempt to log only the header information and inserted text; the message text will be left out.

When you create the log message file, you can specify where in the message the additional data is to be inserted. Further information is provided below.

The data for **msg_ins_ptr** consists of a series of up to nine null-terminated strings. (Because IBM OS/2 ES version 1.0 supports only three data strings, you may want to limit the inserted text to three strings to ensure compatibility.)

Creating a Message File

If you want to create your own message file, you must use the utility MKMSGF.

The first three characters of the message number must match the three-character name of the log message file. These three characters are declared at the top of the file as well.

The system finds the message file as follows:

- If you use your own message file, the system assumes the file is in the same directory as your programs executable file.
- If you use the default message file, COM.MSG, the system finds the file automatically, provided the SnaBase for Microsoft Host Integration Server 2009 is loaded.
- If you use the default message file without loading the previously-mentioned software, the system expects DPATH to indicate the path to the message file. This applies only to the Windows version 3.*x* and OS/2 operating systems.

TRANSFER_MS_DATA

The **TRANSFER_MS_DATA** verb builds an SNA request unit containing Network Management Vector Transport (NMVT) data. The verb can send the NMVT data to NetView for centralized problem diagnosis and resolution. The data is logged in the local audit file.

The following structure describes the verb control block (VCB) used by the **TRANSFER_MS_DATA** verb.

Syntax

```
struct transfer_ms_data {
    unsigned short       opcode;
    unsigned char        data_type;
    unsigned char        reserv2;
    unsigned short       primary_rc;
    unsigned long        secondary_rc;
    unsigned char        options;
    unsigned char        reserv3;
    unsigned char        orignтr_id[8];
    unsigned short       dlen;
    unsigned char FAR *  dptr;
};
```

Members

opcode

Supplied parameter. The verb identifying the operation code, SV_TRANSFER_MS_DATA.

data_type

Supplied parameter. Specifies the type of data provided by this verb:

- Use SV_NMVT to generate an NMVT (including the NS header, the major network management vector, and subvectors).
- Use SV_ALERT_SUBVECTORS to generate an RU containing data for an alert in the appropriate format, without the NS header or major NMVT vector.
- Use SV_PDSTATS_SUBVECTORS to generate an RU containing data for problem determination statistics in the appropriate format, without the NS header or major NMVT vector.
- Use SV_USER_DEFINED to generate user-defined data; this data is recorded in the error log but cannot be sent on the systems services control point-physical unit (SSCP-PU) session on the connection configured for diagnostics.

reserv2

A reserved field.

primary_rc

Returned parameter. Specifies the primary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

secondary_rc

Returned parameter. Specifies the secondary return code set by APPC at the completion of the verb. The valid return codes vary depending on the APPC verb issued. See Return Codes for valid error codes for this verb.

options

Supplied parameter. Specifies the desired options by turning individual bits on or off. (Bits 1, 2, and 3 are ignored if **data_type** is set to SV_USER_DEFINED.) See the Remarks section.

reserv3

A reserved field.

A reserved field.

orignt_id

Supplied parameter. Specifies the name of the component issuing **TRANSFER_MS_DATA**. This parameter is optional. Set it to 0x00 if you want the system to ignore it.

dlen

Supplied parameter. Specifies the length of data to be supplied to this verb. The total length of the data (user-supplied data and any added headers or subvectors) must fit into one RU. The maximum RU length is 512 bytes.

dptr

Supplied parameter. Specifies the address of the data to be sent.

Return Codes

SV_OK

Primary return code; the verb executed successfully.

SV_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

SV_DATA_EXCEEDS_RU_SIZE

Secondary return code; the data to be sent was too long. The length of the user-supplied data plus headers and added subvectors must fit in a single RU that is not more than 512 bytes long.

SV_INVALID_DATA_SEGMENT

Secondary return code; the buffer pointed to by **dptr** was not a readable segment or extended beyond the segment boundary.

SV_INVALID_DATA_TYPE

Secondary return code; the **data_type** parameter contained an invalid value.

SV_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

SV_SSCP_PU_SESSION_NOT_ACTIVE

Secondary return code; the NMVT was not sent; either the SSCP-PU session was not active, the node configured to receive diagnostic information was not active, or no network management connection was configured.

SV_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

SV_INVALID_VERB

Primary return code; the **opcode** parameter did not match the operation code of any verb. No verb executed.

SV_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

SV_UNEXPECTED_DOS_ERROR

Primary return code; one of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.

- A CSV was issued from a message loop that was invoked by another application issuing a Windows **SendMessage** function call, rather than the more common Windows **PostMessage** function call. Verb processing cannot take place.

- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

SV_CANCELLED

Primary return code; this code is returned for an asynchronous verb when it has been shut down by a WinCSVCleanup call.

SV_SERVER_RESOURCE_NOT_FOUND

Primary return code; no communication server was found that could provide the requested function.

SV_SERVER_RESOURCES_LOST

Primary return code; the communications server that was providing the function was lost due to a connection failure.

SV_SERVER_CONN_FAILURE

Secondary return code; the connection to the server was lost due to physical path problems; for example, the server may have been powered off.

SV_THREAD_BLOCKING

Primary return code; this verb exceeds the maximum number of simultaneous synchronous verbs allowed.

Remarks

To specify options, turn bits on or off as follows:

Bit	**Description**
0	TIME_STAMP_SUBVECTOR. Adds date/time subvector to data. Allowed values include SV_ADD and SV_NO_ADD.
1	PRODUCT_SET_ID_SUBVECTOR. Adds Product_Set_ID subvector to data. This allows network management services to identify the sender of an alert. Allowed values include SV_ADD and SV_NO_ADD.
2	SSCP_PU_SESSION. Sends the data on the SSCP-PU session on the connection configured for diagnostics if the session is active. (The data is added to the error log regardless of whether it is sent on the session or whether SV_STATE_CHECK or SV_COMM_SUBSYSTEM_NOT_LOADED is returned.) Allowed values include SV_SEND and SV_NO_SEND.
3	LOCAL_LOGGING. Logs local alerts that are retrieved from the error log and forwarded to the host. This option is valid only when **data_type** SV_NMVT or **data_type** SV_ALERT_SUBVECTORS with option SV_SEND is specified. Allowed values include SV_LOG and SV_NO_LOG.
4 through 7	Reserved

CSV Extensions for the Windows Environment

This section describes API extensions to the Windows® Common Service Verb (CSV) API. Windows CSV allows multithreaded Microsoft® Windows-based processes. Multithreading is the running of several processes in rapid sequence within a single program. A process contains one or more threads of execution. All references to threads in this document refer to actual threads in multithreaded Windows environments.

For each extension, this section provides a definition of the function, syntax, returns, and remarks for using the function.

This section contains:

WinAsyncCSV

The **WinAsyncCSV** function provides an asynchronous entry point for TRANSFER_MS_DATA only. If this function is used for any other verb, the behavior will be synchronous. This function was used instead of the blocking version of the verb under Microsoft Windows version 3.*x*. Windows version 3.x is no longer supported.

HANDLE WINAPI WinAsyncCSV(

Syntax

```
    HWND hWnd,
    long lpVcb
);
```

Parameters

hWnd

Handle of window to receive message.

lpVcb

Pointer to the verb control block.

Return Value

The return value specifies whether the asynchronous resolution request was successful. If the function was successful, the return value is an asynchronous task handle. If the function was not successful, a zero is returned.

Remarks

When the asynchronous operation is complete, the applications window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinAsyncCSV" as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call. The *lParam* argument contains the original VCB pointer and can be dereferenced to determine the final return code.

If the function returns successfully, a "WinAsyncCSV" message will be posted to the application when the operation completes or the conversation is canceled.

WinCSVCleanup

The **WinCSVCleanup** function terminates and deregisters an application from a Windows® CSV implementation.

Syntax

```
BOOL WINAPI WinCSVCleanup(void);
```

Return Value

The return value specifies whether the deregistration was successful. If the value is nonzero, the application was successfully deregistered. The application was not deregistered if a value of zero is returned.

Remarks

Use **WinCSVCleanup** to indicate deregistration of a Windows CSV application from a Windows CSV implementation. This function can be used, for example, to free up resources allocated to the specific application.

WinCSVStartup

The **WinCSVStartup** function allows an application to specify the version of Windows CSV required and to retrieve details of the specific Windows CSV implementation. This function must be called by an application to register itself with a Windows CSV implementation before issuing any further Windows CSV calls.

Syntax

```
int WINAPI WinCSVStartup(
WORD wVersionRequired,
LPWCSVDATA lpwcsvdata
);
```

Parameters

wVersionRequired

Specifies the version of Windows CSV support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number. The current version of the Windows CSV API is 1.0.

lpwcsvdata

A pointer to the CSV data structure. The **CSVDATA** structure is defined as follows:

```
typedef struct tagWCSVDATA {
    WORD wVersion;
    char szDescription[WCSVDESCRIPTION_LEN+1];
}  CSVDATA, FAR * LPWCSVCDATA;
```

where **WCSVDESCRIPTION** is defined to be 127 and the structure members are as follows:

wVersion

The version of Windows CSV supported. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.

szDescription

A description string identifying the vendor of the Windows CSV DLL.

This **CVSDATA** structure provides information about the underlying Windows CSV DLL implementation. The first *wVersion* field has the same structure as the *wVersionRequired* parameter, and the *szDescription* field contains a string identifying the vendor of the Windows CSV DLL. The description field is only meant to provide a display string for the application and should not be used to programmatically distinguish between Windows CSV implementations.

Return Values

The return value specifies whether the application was registered successfully and whether the Windows CSV implementation can support the specified version number. If the value is zero, it was registered successfully. Otherwise, the return value is one of the following:

WCSVSYSNOTREADY

Indicates that the underlying network system is not ready for network communication.

WCSVVERNOTSUPPORTED

The version of Windows CSV support requested is not provided by this particular Windows CSV implementation.

WCSVINVALID

The Windows CSV version specified by the application is not supported by this DLL.

Remarks

To support future Windows CSV implementations and applications that may have functionality differences from Windows CSV version 1.0, a negotiation takes place in **WinCSVStartup**. An application passes to **WinCSVStartup** the Windows CSV version that it can use. If this version is lower than the lowest version supported by the Windows CSV DLL, the DLL cannot support the

application and **WinCSVStartup** fails. If the version is not lower, however, the call succeeds and returns the highest version of Windows CSV supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows CSV DLL on the system.

This negotiation allows both a Windows CSV DLL and a Windows CSV application to support a range of Windows CSV versions. An application can successfully use a DLL if there is any overlap in the versions. The following table illustrates how **WinCSVStartup** works in conjunction with different application and DLL versions.

Application versions	DLL versions	To WinCSVStartup	From WinCSVStartup	Result
1.0	1.0	1.0	1.0	Use 1.0
1.0, 2.0	1.0	2.0	1.0	Use 1.0
1.0	1.0, 2.0	1.0	2.0	Use 1.0
1.0	2.0, 3.0	1.0	WCSVINVALID	Fail
2.0, 3.0	1.0	3.0	1.0	App Fails
1.0, 2.0, 3.0	1.0, 2.0, 3.0	3.0	3.0	Use 3.0

After making its last Windows CSV call, an application should call WinCSVCleanup.

Each Windows CSV implementation must make a **WinCSVStartup** call before issuing any other Windows CSV calls. Consequently, this function can be used for initialization purposes.

Common APPC Return Codes

This section describes the primary and, if applicable, secondary return codes for the Advanced Program-to-Program Communications (APPC) verbs. The return codes are listed in hexadecimal order.

This section contains:

Primary APPC Return Codes

0000
AP_OK

The verb executed successfully.

0001
AP_PARAMETER_CHECK

The verb did not execute because of a parameter error.

0002
AP_STATE_CHECK

The verb did not execute because it was issued in an invalid state.

0003
AP_ALLOCATION_ERROR

APPC failed to allocate a conversation. The conversation state is set to RESET.

This code can be returned through a verb issued after ALLOCATE or MC_ALLOCATE.

0005
AP_DEALLOC_ABEND (for a mapped conversation)

The conversation has been deallocated for one of the following reasons:

- The partner transaction program (TP) issued MC_DEALLOCATE with **dealloc_type** set to AP_ABEND.
- The partner TP encountered an ABEND, causing the partner logical unit (LU) to send an **MC_DEALLOCATE** request.

0006
AP_DEALLOC_ABEND_PROG (for a basic conversation)

The conversation has been deallocated for one of the following reasons:

- The partner TP issued DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG.
- The partner TP encountered an ABEND, causing the partner LU to send a **DEALLOCATE** request.

0007
AP_DEALLOC_ABEND_SVC (for a basic conversation)

The conversation has been deallocated because the partner TP issued DEALLOCATE with **dealloc_type** set to AP_ABEND_SVC.

0008
AP_DEALLOC_ABEND_TIMER (for a basic conversation)

The conversation has been deallocated because the partner TP issued DEALLOCATE with **dealloc_type** set to AP_ABEND_TIMER.

0009
AP_DEALLOC_NORMAL

The partner TP has deallocated the conversation without requesting confirmation.

000C
AP_PROG_ERROR_NO_TRUNC

The partner TP has issued one of the following verbs while the conversation was in SEND state:

- SEND_ERROR with **err_type** set to AP_PROG

- MC_SEND_ERROR

Data was not truncated.

000F
AP_CONV_FAILURE_RETRY

The conversation was terminated because of a temporary error. Restart the TP to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

0010
AP_CONV_FAILURE_NO_RETRY

The conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

0011
AP_SVC_ERROR_NO_TRUNC

While in SEND state, the partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC. Data was not truncated.

0012
AP_PROG_ERROR_TRUNC/AP_SVC_ERROR_TRUNC

In SEND state, after sending an incomplete logical record, the partner TP issued SEND_ERROR. The local TP may have received the first part of the logical record.

0013
AP_SVC_ERROR_PURGING

The partner TP (or partner LU) issued SEND_ERROR with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner TP may have been purged.

0014
AP_UNSUCCESSFUL

No data is immediately available from the partner TP.

0017
AP_CNOS_LOCAL_RACE_REJECT

APPC is currently processing a CNOS verb issued by a local LU.

0018
AP_CNOS_PARTNER_LU_REJECT

The partner LU rejected a CNOS request from the local LU.

0019
AP_CONVERSATION_TYPE_MIXED

The TP has issued both basic and mapped conversation verbs. Only one type can be issued in a single conversation.

0021
AP_CANCELED

The local TP issued one of the following verbs, which canceled RECEIVE_AND_POST or MC_RECEIVE_AND_POST:

- DEALLOCATE with **dealloc_type** set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- MC_DEALLOCATE with **dealloc_type** set to AP_ABEND
- SEND_ERROR or MC_SEND_ERROR
- TP_ENDED

Issuing one of these verbs causes the semaphore to be cleared.

F002
AP_TP_BUSY

The local TP has issued a call to APPC while APPC was processing another call for the same TP. This can occur if the local TP has multiple threads, and more than one thread is issuing APPC calls using the same **tp_id**.

F003
AP_COMM_SUBSYSTEM_ABENDED

Indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the TP and the PU 2.1 node has been broken (a LAN error).
- The SnaBase at the TP's computer encountered an ABEND.

The system administrator should examine the error log to determine the reason for the ABEND.

F004
AP_COMM_SUBSYSTEM_NOT_LOADED

A required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

F005
AP_CONV_BUSY

There can only be one outstanding conversation verb at a time on any conversation.

F006
AP_THREAD_BLOCKING

The calling thread is already in a blocking call.

F008
AP_INVALID_VERB_SEGMENT

The verb control block (VCB) extended beyond the end of the data segment.

F011
AP_UNEXPECTED_DOS_ERROR

The operating system returned an error to APPC while processing an APPC call from the local TP. The operating system return code is returned through the **secondary_rc**. It appears in Intel byte-swapped order. If the problem persists, consult the system administrator.

F015
AP_STACK_TOO_SMALL

The stack size of the application is too small to execute the verb. Increase the stack size of your application.

F020
AP_INVALID_KEY

The supplied **key** was incorrect.

Secondary APPC Return Codes

00000000
AP_CNOS_ACCEPTED

APPC accepts the session lines and responsibility as specified.

00000001
AP_BAD_TP_ID

The value of **tp_id** did not match a transaction program (TP) identifier assigned by APPC.

00000002
AP_BAD_CONV_ID

The value of **conv_id** did not match a conversation identifier assigned by APPC.

00000003
AP_BAD_LU_ALIAS

APPC cannot find the specified **lu_alias** among those defined.

000000C4
AP_RCV_IMMD_BAD_FILL (for a basic conversation)

The **fill** parameter was set to an invalid value.

00000004
AP_ALLOCATION_FAILURE_NO_RETRY

The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

00000005
AP_ALLOCATION_FAILURE_RETRY

The conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

00000006
AP_INVALID_DATA_SEGMENT

The program initiation parameters (PIP) data was longer than the allocated data segment, or the address of the PIP data buffer was wrong.

00000007
AP_CNOS_NEGOTIATED

APPC accepts the session limits and responsibility as negotiable by the partner logical unit (LU). Values that can be negotiated are:

plu_mode_session_limit

min_conwinners_source

min_conwinners_target

responsible

drain_target

000000D7
AP_BAD_RETURN_STATUS_WITH_DATA

The specified **rtn_status** value was not recognized by APPC.

00000011
AP_BAD_CONV_TYPE (for a basic conversation)

The value specified for **conv_type** was invalid.

00000012
AP_BAD_SYNC_LEVEL

The value specified for **sync_level** was invalid.

00000013
AP_BAD_SECURITY

The value specified for **security** was invalid.

00000014
AP_BAD_RETURN_CONTROL

The value specified for **rtn_ctl** was invalid.

00000016
AP_PIP_LEN_INCORRECT

The value of **pip_dlen** was greater than 32767.

00000017
AP_NO_USE_OF_SNASVCMG (for a mapped conversation)

SNASVCMG is not a valid value for **mode_name**.

00000018
AP_UNKNOWN_PARTNER_MODE

The value specified for **mode_name** was invalid.

00000031
AP_CONFIRM_ON_SYNC_LEVEL_NONE

The local TP attempted to use CONFIRM or MC_CONFIRM in a conversation with a synchronization level of AP_NONE. The synchronization level, established by ALLOCATE or MC_ALLOCATE, must be AP_CONFIRM_SYNC_LEVEL.

00000032
AP_CONFIRM_BAD_STATE

The conversation was not in SEND state.

00000033
AP_CONFIRM_NOT_LL_BDY

The conversation for the local TP was in SEND state, and the local TP did not finish sending a logical record.

00000051
AP_DEALLOC_BAD_TYPE

The **dealloc_type** parameter was not set to a valid value.

00000052
AP_DEALLOC_FLUSH_BAD_STATE

The conversation was not in SEND state and the TP attempted to flush the send buffer. This attempt occurred because the value of **dealloc_type** was AP_FLUSH or because the value of **dealloc_type** was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_NONE. In either case, the conversation must be in SEND state.

00000053
AP_DEALLOC_CONFIRM_BAD_STATE

The conversation was not in SEND state, and the TP attempted to flush the send buffer and send a confirmation request.

00000055
AP_DEALLOC_NOT_LL_BDY (for a basic conversation)

The conversation was in SEND state, and the TP did not finish sending a logical record. The **dealloc_type** parameter was set to AP_SYNC_LEVEL or AP_FLUSH.

00000057
AP_DEALLOC_LOG_LL_WRONG

The LL field of the general data stream (GDS) error log variable did not match the actual length of the log data.

00000061
AP_FLUSH_NOT_SEND_STATE

The conversation was not in SEND state.

000000A1
AP_P_TO_R_INVALID_TYPE

The **ptr_type** parameter was not set to a valid value.

000000A2
AP_P_TO_R_NOT_LL_BDY

The local TP did not finish sending a logical record.

000000A3
AP_P_TO_R_NOT_SEND_STATE

The conversation was not in SEND state.

000000B1
AP_RCV_AND_WAIT_BAD_STATE

The conversation was not in RECEIVE or SEND state when the TP issued this verb.

000000B2
AP_RCV_AND_WAIT_NOT_LL_BDY (for a basic conversation)

The conversation was in SEND state; the TP began but did not finish sending a logical record.

000000B5
AP_RCV_AND_WAIT_BAD_FILL (for a basic conversation)

The **fill** parameter was set to an invalid value.

000000C1
AP_RCV_IMMD_BAD_STATE

The conversation was not in RECEIVE state.

000000D1
AP_RCV_AND_POST_BAD_STATE

The conversation was not in RECEIVE or SEND state when the TP issued this verb.

000000D2
AP_RCV_AND_POST_NOT_LL_BDY

The conversation was in SEND state; the TP began but did not finish sending a logical record.

000000D5
AP_RCV_AND_POST_BAD_FILL

The **fill** parameter was set to an invalid value.

000000D6
AP_INVALID_SEMAPHORE_HANDLE

The address of the RAM semaphore or system semaphore handle was invalid.

Note
APPC cannot trap all invalid semaphore handles. If the TP passes a bad RAM semaphore handle, a protection violation results.

000000D7
AP_BAD_RETURN_STATUS_WITH_DATA

The specified **rtn_status** value was not recognized by APPC.

000000E1
AP_R_T_S_BAD_STATE

The conversation is not in an allowed state when the TP issued this verb.

000000F1
AP_BAD_LL (for a basic conversation)

The logical record length field of a logical record contained an invalid value — 0x0000, 0x0001, 0x8000, or 0x8001. See About Transaction Programs for information on logical records.

000000F2
AP_SEND_DATA_NOT_SEND_STATE

The local TP issued SEND_DATA or MC_SEND_DATA, but the conversation was not in SEND state.

000000F5
AP_SEND_DATA_CONFIRM_ON_SYNC_NONE

The type CONFIRM is not permitted for a conversation that was allocated with a **sync_level** of NONE.

000000F6
AP_SEND_DATA_NOT_LL_BDY (for a basic conversation)

The TP started but did not finish sending a logical record. This occurs only when **type** is one of the following:

AP_SEND_DATA_CONFIRM

AP_SEND_DATA_DEALLOC_FLUSH

AP_SEND_DATA_DEALLOC_SYNC_LEVEL

AP_SEND_DATA_P_TO_R_FLUSH

AP_SEND_DATA_P_TO_R_SYNC_LEVEL

00000102
AP_SEND_ERROR_LOG_LL_WRONG (for a basic conversation)

The LL field of the error log GDS variable did not match the actual length of the data.

00000103
AP_SEND_ERROR_BAD_TYPE (for a basic conversation)

The value of **err_type** was invalid.

00000105
AP_BAD_ERROR_DIRECTION

The specified **err_dir** was not recognized by APPC.

00000150
AP_CNOS_IMPLICIT_PARALLEL

APPC does not permit a program to change the session limit for a mode other than SNASVCMG mode for the implicit partner template when the template specifies parallel sessions. (The term "template" is used because many of the actual values are yet to be filled in.)

00000151
AP_CANT_RAISE_LIMITS

APPC does not permit setting session limits to a nonzero value unless the limits currently are zero.

00000152
AP_AUTOACT_EXCEEDS_SESSLIM

On the CNOS verb, the value for **auto_activate** is greater than the value for **partner_lu_mode_session_limit**.

00000153
AP_ALL_MODE_MUST_RESET

APPC does not permit a nonzero session limit when **mode_name_select** indicates ALL.

00000154
AP_BAD_SNASVCMG_LIMITS

Your program specified invalid settings for the **partner_lu_mode_session_limit**, **min_conwinners_source**, or

min_conwinners_target parameters when **mode_name** was supplied.

00000155
AP_MIN_GT_TOTAL

The sum of **min_conwinners_source** and **min_conwinners_target** specifies a number greater than **partner_lu_mode_session_limit**.

00000156
AP_MODE_CLOSED

The local LU cannot negotiate a nonzero session limit because the local maximum session limit at the partner LU is zero.

00000156
AP_CNOS_MODE_CLOSED

The local LU cannot negotiate a nonzero session limit because the local maximum session limit at the partner LU is zero.

00000157
AP_CNOS_MODE_NAME_REJECT

The partner LU does not recognize the specified mode name.

00000159
AP_RESET_SNA_DRAINS

The SNASVCMG mode does not support the **drain** parameter values.

0000015A
AP_SINGLE_NOT_SRC_RESP

For a single-session CNOS verb, APPC permits only the local (source) LU to be responsible for deactivating sessions.

0000015B
AP_BAD_PARTNER_LU_ALIAS

APPC did not recognize the supplied **partner_lu_alias**.

0000015C
AP_EXCEEDS_MAX_ALLOWED

Your program issued a CNOS verb, specifying a **partner_lu_mode_session_limit** number and **set_negotiable** (NO).

0000015D
AP_CHANGE_SRC_DRAINS

APPC does not permit **mode_name_select** (ONE) and **drain_source** (YES) when **drain_source** (NO) is currently in effect for the specified mode.

0000015E
AP_LU_DETACHED

A command reset the definition of the local LU before the CNOS verb tried to specify the LU.

0000015F
AP_CNOS_COMMAND_RACE_REJECT

The local LU is currently processing a CNOS verb issued by the partner LU.

00000167
AP_SNASVCMG_RESET_NOT_ALLOWED

Your local program attempted to issue the CNOS verbs for the mode named SNASVCMG, specifying a session limit of zero.

000001B4
AP_DISPLAY_INFO_EXCEEDS_LENGTH

The returned DISPLAY information did not fit in the buffer.

000001B5
DISPLAY_INVALID_CONSTANT

The value supplied for NUM_SECTIONS or INIT_SEC_LEN is invalid.

00000506
AP_UNDEFINED_TP_NAME

In the configuration file for your application, APPC could not find an invokable TP name matching the value of **tp_name**.

00000509
AP_ALLOCATE_NOT_PENDING

APPC did not find an incoming allocate (from the invoking TP) to match the value of **tp_name**, supplied by RECEIVE_ALLOCATE. **RECEIVE_ALLOCATE** waited for the incoming allocate and eventually timed out.

00000519
AP_CPSVCMG_MODE_NOT_ALLOWED

The mode named CPSVCMG cannot be specified as the **mode_name** on the deactivate session verb.

00000525
AP_INVALID_PROCESS

The process issuing RECEIVE_ALLOCATE was different from the one started by APPC.

080F6051
AP_SECURITY_NOT_VALID

The user identifier or password specified in the allocation request was not accepted by the partner LU.

084B6031
AP_TRANS_PGM_NOT_AVAIL_RETRY

The remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition may be temporary, such as a time-out. The reason for the error may be logged on the remote node. Retry the allocation.

084C0000
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY

The remote LU rejected the allocation request because it was unable to start the requested partner TP. The condition is permanent. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

10086021
AP_TP_NAME_NOT_RECOGNIZED

The partner LU does not recognize the TP name specified in the allocation request.

10086031
AP_PIP_NOT_ALLOWED

The allocation request specified PIP data, but either the partner TP does not require this data, or the partner LU does not support it.

10086032
AP_PIP_NOT_SPECIFIED_CORRECTLY

The partner TP requires PIP data, but the allocation request specified either no PIP data or an incorrect number of parameters.

10086034
AP_CONVERSATION_TYPE_MISMATCH

The partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

10086041
AP_SYNC_LEVEL_NOT_SUPPORTED

The partner TP does not support the **sync_level** (AP_NONE or AP_CONFIRM_SYNC_LEVEL) specified in the allocation request, or the **sync_level** was not recognized.

Common CSV Return Codes

This section describes the primary and, if applicable, secondary return codes for the Microsoft® Windows® Common Service Verb (CSV) API. The return codes are listed in hexadecimal order.

In This Section

- Primary CSV Return Codes
- Secondary CSV Return Codes

Primary CSV Return Codes

0000
SV_OK

The verb executed successfully.

0001
SV_PARAMETER_CHECK

The verb did not execute because of a parameter error.

0002
SV_STATE_CHECK

The verb did not execute because it was issued in an invalid state.

0021
SV_CANCELLED

This code is returned for an asynchronous verb when it has been shut down by a WinCSVCleanup call.

0030
SV_FILE_ALREADY_EXISTS

When the SV_NEW file option was used, the file name specified was the name of an existing file.

0031
SV_OUTPUT_DEVICE_FULL

There is insufficient space on the device where the output file resides. Retry the operation after freeing additional disk space.

F006
SV_THREAD_BLOCKING

This verb exceeds the maximum number of simultaneous synchronous verbs allowed.

F008
SV_INVALID_VERB_SEGMENT

The verb control block (VCB) extended beyond the end of the data segment.

F011
SV_UNEXPECTED_DOS_ERROR

One of the following conditions occurred:

- The Microsoft® Windows® 2000 system encountered an error while processing the verb. The operating system return code was returned through the secondary return code. If the problem persists, contact the system administrator for corrective action.
- A CSV was issued from a message loop that was invoked by another application issuing a Windows environment **SendMessage** function call, rather than the more common Windows environment **PostMessage** function call. Verb processing cannot take place.
- A CSV was issued when **SendMessage** invoked your application. You can determine whether your application has been invoked with **SendMessage** by using the **InSendMessage** Windows API function call.

F012
SV_COMM_SUBSYSTEM_NOT_LOADED

A required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

F024
SV_SERVER_RESOURCE_NOT_FOUND

No communication server was found that could provide the requested function.

F026
SV_SERVER_RESOURCE_LOST

The communications server that was providing the function was lost due to a connection failure.

FFFF
SV_INVALID_VERB

The **opcode** parameter did not match the operation code of any verb. No verb executed.

Secondary CSV Return Codes

00000006
SV_INVALID_DATA_SEGMENT

The data buffer containing the source or target string did not fit in one segment, or the target segment was not a read/write segment. This applies only to the Microsoft® Windows® and OS/2 systems.

00000301
SV_SSCP_PU_SESSION_NOT_ACTIVE

The Network Management Vector Transport (NMVT) was not sent; either the system services control point-physical unit (SSCP-PU) session was not active, the node configured to receive diagnostic information was not active, or no network management connection was configured.

00000302
SV_DATA_EXCEEDS_RU_SIZE

The data to be sent was too long. The length of the user-supplied data plus headers and added subvectors must fit in a single request unit (RU) that is not more than 512 bytes long.

00000303
SV_INVALID_DATA_TYPE

The **data_type** parameter contained an invalid value.

00000401
SV_INVALID_DIRECTION

The **direction** parameter contained an invalid value.

00000402
SV_INVALID_CHARACTER_SET

The **char_set** parameter contained an invalid value.

00000404
SV_INVALID_FIRST_CHARACTER

The first character of a type A source string was invalid.

00000405
SV_TABLE_ERROR

One of the following occurred:

- The file containing the user-written type G conversion table was not specified by the environment variable CSVTBLG.
- The table was not in the correct format.
- The file specified by the CSVTBLG variable was not found.

00000406
SV_CONVERSION_ERROR

One or more characters in the source string were not found in the conversion table. These characters were converted to nulls (0x00). The verb still executed.

00000621
SV_INVALID_MESSAGE_ACTION

The **msg_act** parameter contained an invalid value.

00000624
SV_INVALID_SET

The **dt_set** parameter contained an invalid value.

00000629
SV_COPY_TRACE_IN_PROGRESS

A previously issued COPY_TRACE_TO_FILE is still in progress.

0000062A
SV_TRACE_NOT_STOPPED

A trace was in progress when the verb was issued.

0000062B
SV_INVALID_FILE_OPTION

A value other than SV_NEW or SV_OVERWRITE was specified for **file_option**.

0000062C
SV_TRACE_BUFFER_EMPTY

The trace storage buffer did not contain any data.

0000062F
SV_INVALID_RESET_TRACE

The **reset_trc** parameter contained an invalid value.

00000630
SV_INVALID_CHAR_NOT_FOUND

The **char_not_fnd** parameter contained an invalid value.

00000631
SV_INVALID_SOURCE_CODE_PAGE

The code page specified by **source_cp** is not supported.

00000632
SV_INVALID_TARGET_CODE_PAGE

The code page specified by **target_cp** is not supported.

030000AB
SV_SERVER_COMM_FAILURE

The connection to the server was lost due to physical path problems; for example, the server may have been powered off.

CPI-C Programmer's Reference

This section of the Host Integration Server 2009 Developer's Guide provides information about the calls, extensions, and return codes that make up the CPI-C.

For general information about programming for CPI-C, see the CPI-C Programmer's Guide section of the SDK.

For sample code that uses CPI-C, see CPI-C Samples.

In This Section

CPI-C Calls

Extensions for the Windows Environment

CPI-C Common Return Codes

CPI-C Calls

This section describes the Common Programming Interface for Communications (CPI-C) calls. The following information is supplied for each call:

- The pseudonym for the call and the actual C function name.
- A definition of the call.
- A list of the parameters used by the call and the data type for each parameter. The prototype of each function is declared in the WINCPIC.H file.
- A description of each input and output parameter. The parameter names are pseudonyms and the actual names for these parameters are declared by the application program. The description includes the possible values of the parameter.
- The conversation states in which the call can be issued.
- The states to which the conversation can change upon return from the call. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the use of the call.

Data Types

The data types for the parameters supplied to and received from CPI-C are established as symbolic constants by **#define** statements in the WINCPIC.H file. For example, CM_INT32 represents **signed long int** and CM_PTR represents **far ***. Using symbolic constants improves the portability of CPI-C applications.

For ease of understanding, this reference presents the data types in absolute (not **#defined**) terms.

In writing applications, you should use the symbolic constants from the WINCPIC.H file.

Symbolic Constants

Most parameters supplied to and returned by CPI-C are 32-bit integers. To simplify coding, the values for these parameters are represented by meaningful symbolic constants, which are established by **#define** statements in the WINCPIC.H header file. For example, the value CM_MAPPED_CONVERSATION represents the integer 1. For the sake of readability, use only the symbolic constants when writing programs.

Strings

All strings are in ASCII format when passed across the CPI-C interface.

Validity of output parameters

The parameters returned by CPI-C are valid only if the CPI-C call is executed successfully, as indicated by a return code of CM_OK.

In This Section

- Convert_Outgoing
- Deallocate
- Delete_CPIC_Side_Information
- Extract_Conversation_Security_Type
- Extract_Conversation_Security_User_ID
- Extract_Conversation_State
- Extract_Conversation_Type
- Extract_CPIC_Side_Information
- Extract_Mode_Name
- Extract_Partner_LU_Name
- Extract_Sync_Level
- Extract_TP_Name
- Flush
- Initialize_Conversation
- Prepare_To_Receive
- Receive
- Request_To_Send
- Send_Data
- Send_Error
- Set_Conversation_Security_Password
- Set_Conversation_Security_Type
- Set_Conversation_Security_User_ID
- Set_Conversation_Type
- Set_CPIC_Side_Information
- Set_Deallocate_Type
- Set_Error_Direction
- Set_Fill

- Set_Log_Data
- Set_Mode_Name
- Set_Partner_LU_Name
- Set_Prepare_To_Receive_Type
- Set_Processing_Mode
- Set_Receive_Type
- Set_Return_Control
- Set_Send_Type
- Set_Sync_Level
- Set_TP_Name
- Specify_Local_TP_Name
- Specify_Windows_Handle
- Test_Request_To_Send_Received
- Wait_For_Conversation
- CPI-C Functions Not Supported

Accept_Conversation

The **Accept_Conversation** call (function name **cmaccp**) is issued by the invoked program to accept the incoming conversation and set certain conversation characteristics. For a list of initial conversation characteristics, see Initial Conversation Characteristics.

Syntax

```
CM_ENTRY Accept_Conversation(
   unsigned char FAR *conversation_ID,
      CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Returned parameter. Specifies the identifier for the conversation. It is used by subsequent CPI-C calls and is returned if the return code is either CM_OK or CM_OPERATION_INCOMPLETE. If the return code is CM_OPERATION_INCOMPLETE, the *conversation_ID* parameter can be used by the application to wait for or cancel the conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; there is no incoming conversation (blocking mode only), or no local transaction program (TP) name has been set up.

CM_OPERATION_INCOMPLETE

Primary return code; a nonblocking operation has been started on the conversation but is not complete. The program can issue Wait_For_Conversation to wait for the operation to complete or Cancel_Conversation to cancel the operation and conversation.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in RESET state when **Accept_Conversation** is issued.

If the call is successful, the conversation changes to RECEIVE state. If the call fails, the state remains unchanged.

Remarks

Upon successful execution of this call, CPI-C generates an 8-byte conversation identifier. This identifier is a required parameter for all other CPI-C calls issued by the invoked program on this conversation.

Incoming conversations will be accepted according to the target TP name that they specify, which must match local TP names that have been set up. Local TP names can be set up by implementation-dependent methods, or by the program calling Specify_Local_TP_Name. In this way, a program can have more than one local TP name. The program can call Extract_TP_Name to discover the name specified on the incoming conversation.

The operation is performed in nonblocking mode if the program has called **Specify_Local_TP_Name** previously; otherwise it is performed in blocking mode.

Allocate

The **Allocate** call (function name **cmallc**) is issued by the invoking program to allocate a conversation with the partner program, using the current conversation characteristics. CPI-C can also allocate a session between the local logical unit (LU) and partner LU if one does not already exist.

Syntax

```
CM_ENTRY Allocate(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the conversation identifier. The value of this parameter was returned by Initialize_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; this value indicates that a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; a nonblocking operation has been started on the conversation but is not complete. The program can issue Wait_For_Conversation to wait for the operation to complete or Cancel_Conversation to cancel the operation and conversation.

CM_PARAMETER_ERROR

Primary return code; one of the following occurred:

- The mode name derived from the side information or set by Set_Mode_Name is not valid.
- The mode name is used by SNA service transaction programs (TPs); the invoking program does not have the authority to use this mode name. An example is SNASVCMG.
- The partner program derived from the side information is an SNA service TP; the local program does not have the privilege required to allocate a conversation to an SNA service TP.
- The partner program is a service TP, which participates in basic conversations, but the conversation is set to CM_MAPPED_CONVERSATION.
- The partner LU name derived from the side information or set by Set_Partner_LU_Name is not valid.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is not valid, or the address of a variable is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_UNSUCCESSFUL

Primary return code; the conversations return-control characteristic is set to CM_IMMEDIATE and the local LU did not have an available contention-winner session.

The following return codes can be generated if the conversations return-control type is set to CM_WHEN_SESSION_ALLOCATED.

CM_ALLOCATE_FAILURE_NO_RETRY

Primary return code; the conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

CM_ALLOCATE_FAILURE_RETRY

Primary return code; the conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

State Changes

The conversation must be in INITIALIZE state when **Allocate** is issued.

State changes, summarized in the following table, are based on the value of the *return_code* parameter.

return_code	**New state**
CM_OK	SEND
CM_ALLOCATE_FAILURE_NO_RETRY	RESET
CM_ALLOCATE_FAILURE_RETRY	RESET
All others	No change

Remarks

The type of conversation allocated is based on the conversation type characteristic: mapped or basic.

When the conversation has been allocated by this call, the following conversation characteristics cannot be changed:

- Conversation type
- Mode name
- Partner LU name
- Partner program name
- Return control
- Synchronization level
- Conversation security
- User identifier
- Password

To send the allocation request immediately, the invoking program can issue Flush or Confirm immediately after **Allocate**. Otherwise, the allocate request accumulates with other data in the local LUs send buffer until the buffer is full.

By issuing **Confirm** after **Allocate**, the invoking program can immediately determine whether the allocation was successful (if

the conversation synchronization level is set to CM_CONFIRM).

If the partner LU rejects the allocation request generated by **Allocate**, the error is returned to the invoking program on a subsequent call.

Cancel_Conversation

The **Cancel_Conversation** call (function name **cmcanc**) cancels any outstanding operation on a conversation (an operation returned with CM_OPERATION_INCOMPLETE) and the conversation itself.

Syntax

```
CM_ENTRY Cancel_Conversation(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Returned parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in any state except RESET.

When the return code is CM_OK, the conversation state becomes RESET.

Remarks

Cancel_Conversation can be called while another operation is active for the specified *conversation_ID*. This allows an application to end any CPI-C action, but will terminate the conversation. This call can be issued regardless of the current application processing mode. Any outstanding operations will return with CM_DEALLOCATED_ABEND as the return code.

The conversation is terminated by a Deallocate with *deallocate_type* set to ABEND_SVC. No *log_data* is sent. The system may be unable to do this immediately, but any delay is transparent to the program.

Note

If **Cancel_Conversation** is called while there are outstanding Specify_Windows_Handle asynchronous calls, these calls are canceled. The return codes are set to canceled, and a completion message is posted.

Confirm

The **Confirm** call (function name **cmcfm**) sends the contents of the send buffer of the local logical unit (LU) and a confirmation request to the partner program and waits for confirmation. For Microsoft® Windows Server™ 2003 and Windows® 2000, run a background thread for all CPI-C communications and preserve the foreground thread for user interface only.

Syntax

```
CM_ENTRY Confirm(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *request_to_send_received,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

request_to_send_received

Returned parameter. Provides the request-to-send-received indicator. Possible values are:

CM_REQ_TO_SEND_RECEIVED

The partner program issued Request_To_Send, which requests the local program to change the conversation to RECEIVE state.

CM_REQ_TO_SEND_NOT_RECEIVED

The partner program did not issue **Request_To_Send**. This value is not relevant if *return_code* is set to CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully. The partner program issued the Confirmed call.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Windows message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The local program attempted to use **Confirm** in a conversation with a synchronization level of CM_NONE. The synchronization level must be CM_CONFIRM.

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The conversation was not in SEND or SEND_PENDING state.
- The basic conversation for the local program was in SEND state, and the local program did not finish sending a logical record.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner LU or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request is not accepted by the partner LU.

CM_SYNC LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued Send_Error. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameter set to CM_DEALLOCATE_ABEND. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.

- The partner program terminated normally but did not deallocate the conversation before terminating.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.

- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued Send_Error with the type parameter set to SVC. Data sent to the partner program may have been purged.

State Changes

The conversation can be in SEND or SEND_PENDING state when **Confirm** is issued.

State changes, summarized in the following table, are based on the value of the *return_code* parameter.

return_code	**New state**
CM_OK	No change
Call was issued in SEND state	No change
Call was issued in SEND_PENDING state	SEND
CM_PROGRAM_ERROR_PURGING	RECEIVE
CM_SVC_ERROR_PURGING	RECEIVE
CM_CONVERSATION_TYPE_MISMATCH	RESET
CM_PIP_NOT_SPECIFIED_CORRECTLY	RESET
CM_SECURITY_NOT_VALID	RESET
CM_SYNC_LEVEL_NOT_SUPPORTED_PGM	RESET
CM_TPN_NOT_RECOGNIZED	RESET
CM_TP_NOT_AVAILABLE_NO_RETRY	RESET
CM_TP_NOT_AVAILABLE_RETRY	RESET

CM_RESOURCE_FAILURE_NO_RETRY	RESET
CM_RESOURCE_FAILURE_RETRY	RESET
CM_DEALLOCATED_ABEND	RESET
CM_DEALLOCATED_ABEND_SVC	RESET
CM_DEALLOCATED_ABEND_TIMER	RESET
All others	No change

Remarks

In response to **Confirm**, the partner program normally issues Confirmed to confirm that it has received the data without error. (If the partner program encounters an error, it issues Send_Error or uses Deallocate to abnormally deallocate the conversation.)

The program can issue **Confirm** only if the conversations synchronization level is CM_CONFIRM.

Confirm waits for a response from the partner program. A response is generated by one of the following CPI-C calls in the partner program:

- Confirmed
- Send_Error
- 5Deallocate with the conversations deallocate type set to CM_DEALLOCATE_ABEND

Confirmed

The **Confirmed** call (function name **cmcfmd**) replies to a confirmation request from the partner program. It informs the partner program that the local program has not detected an error in the received data. Because the program issuing the confirmation request waits for a confirmation, **Confirmed** synchronizes the processing of the two programs.

Syntax

```
CM_ENTRY Confirmed(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation was not in CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state when the program issued this call.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in one of the following states when the program issues **Confirmed**:

- CONFIRM
- CONFIRM_SEND
- CONFIRM_DEALLOCATE

The new state is determined by the old state—the state of the conversation when the local program issued **Confirmed**. The old state is indicated by the *status_received* value of the preceding Receive call. The following table summarizes the possible state changes when *return_code* is set to CM_OK.

Old state	New state

CONFIRM	RECEIVE
CONFIRM_SEND	SEND
CONFIRM_DEALLOCATE	RESET

Other return codes result in no state change.

Remarks

A confirmation request is issued by one of the following calls in the partner program:

- Confirm.
- Prepare_To_Receive if the prepare-to-receive type is set to CM_PREP_TO_RECEIVE_CONFIRM or to CM_PREP_TO_RECEIVE_SYNC_LEVEL and the conversations synchronization level is set to CM_CONFIRM.
- Deallocate if the deallocate type is set to CM_DEALLOCATE_CONFIRM or to CM_DEALLOCATE_SYNC_LEVEL and the conversations synchronization level is set to CM_CONFIRM.
- Send_Data under the following circumstances:
    - The send type is set to CM_SEND_AND_CONFIRM.
    - The send type is set to CM_SEND_AND_PREPARE_TO_RECEIVE and the prepare-to-receive type is set to CM_PREPARE_TO_RECEIVE_CONFIRM.
    - The send type is set to CM_SEND_AND_PREPARE_TO_RECEIVE, the prepare-to-receive type is set to CM_PREPARE_TO_RECEIVE_SYNC_LEVEL, and the synchronization level is set to CM_CONFIRM.
    - The send type is set to CM_SEND_AND_DEALLOCATE and the deallocate type is set to CM_DEALLOCATE_CONFIRM.
    - The send type is set to CM_SEND_AND_DEALLOCATE, the deallocate type is set to CM_DEALLOCATE_SYNC_LEVEL, and the synchronization level is set to CM_CONFIRM.

A confirmation request is received by the local program through the *status_received* parameter of Receive. The local program can issue **Confirmed** only if the *status_received* parameter is set to one of the following values:

- CM_CONFIRM_RECEIVED
- CM_CONFIRM_SEND_RECEIVED
- CM_CONFIRM_DEALLOC_RECEIVED

Convert_Incoming

The **Convert_Incoming** call (function name **cmcnvi**) converts a string of EBCDIC characters into ASCII. Note that the return conversion can be performed using **Convert_Outgoing**.

Syntax

```
CM_ENTRY Convert_Incoming(
  unsigned char FAR *string,
  CM_INT32 FAR *string_length,
    CM_INT32 FAR *return_code
);
```

Parameters

string

Supplied parameter. Specifies the EBCDIC string to be converted. The string may contain any of the following characters:

- Uppercase A–Z
- Lowercase a–z
- Numbers 0–9
- The period (.)
- Space characters
- The special characters < > + - ( ) & * ; : , ' ? / _ = ".

string_length characters of this string will be replaced by ASCII equivalents.

string_length

Supplied parameter. Specifies the number of characters to be converted (1–32767).

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully, and the *string* parameter now contains the converted ASCII string.

CM_OPERATION_NOT_ACCEPTED

Primary return code; the *string_length* parameter specified an invalid value.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state.

There is no state change.

Remarks

When data is being received in buffer format in a basic conversation, the data buffer may contain multiple logical records, each consisting of a 2-byte length field (NN) followed by the data. The application must extract and convert each data string separately (excluding the length field value). The applications must not attempt to convert the whole buffer in one operation, because this will make the length field values invalid.

Convert_Outgoing

The **Convert_Outgoing** call (function name **cmcnvo**) converts a string of ASCII characters into EBCDIC. Note that the return conversion can be performed using **Convert_Incoming**.

Syntax

```
CM_ENTRY Convert_Outgoing(
    unsigned char FAR *string,
    CM_INT32 FAR *string_length,
    CM_INT32 FAR *return_code
);
```

Parameters

string

Supplied parameter. Specifies the ASCII string to be converted. The string may contain any of the following characters:

- Uppercase A–Z
- Lowercase a–z
- Numbers 0–9
- The period (.)
- Space characters
- The special characters < > + - ( ) & * ; : , ' ? / _ = ".

string_length characters of this string will be replaced by EBCDIC equivalents.

string_length

Supplied parameter. Specifies the number of characters to be converted (1–32767).

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully and the *string* parameter now contains the converted EBCDIC string.

CM_OPERATION_NOT_ACCEPTED

Primary return code; the *string_length* parameter specified an invalid value.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state.

There is no state change.

Remarks

When data is being received in buffer format in a basic conversation, the data buffer may contain multiple logical records, each consisting of a two-byte length field (NN) followed by the data. The application must extract and convert each data string separately (excluding the length field value). The applications must not attempt to convert the whole buffer in one operation, because this will make the length field values invalid.

Deallocate

The **Deallocate** call (function name **cmdeal**) deallocates a conversation between two programs.

Syntax

```
CM_ENTRY Deallocate(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully; the conversation is deallocated.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the following state errors can occur when the deallocate type indicates a normal deallocation (CM_DEALLOCATE_SYNC_LEVEL, CM_DEALLOCATE_FLUSH, CM_DEALLOCATE_CONFIRM):

- The conversation is not in SEND or SEND_PENDING state.
- For a basic conversation, the conversation is in SEND state, but the program did not finish sending a logical record.

The following return codes can be returned when the *deallocate_type* is set to CM_DEALLOCATE_CONFIRM or to CM_DEALLOCATE_SYNC_LEVEL and the conversations synchronization level is set to CM_CONFIRM.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner logical unit (LU) or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

CM_SYNC LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued Send_Error. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued **Deallocate** with the type parameter set to CM_DEALLOCATE_ABEND, or the remote LU did so because of a remote program abnormal-ending condition. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.
- The remote TP terminated normally but did not deallocate the conversation before terminating. Node services at the remote LU deallocated the conversation on behalf of the remote TP.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.
- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the

local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued **Send_Error** with the type parameter set to SVC. Data sent to the partner program may have been purged.

State Changes

Depending on the value of the conversations deallocate type parameter (set by Set_Deallocate_Type), the conversation can be in one of the states indicated in the following table when the program issues **Deallocate**:

Deallocate type	**Allowed state**
CM_DEALLOCATE_FLUSH	SEND or SEND_PENDING
CM_DEALLOCATE_CONFIRM	SEND or SEND_PENDING
CM_DEALLOCATE_SYNC_LEVEL	SEND or SEND_PENDING
CM_DEALLOCATE_ABEND	Any except RESET

State changes, summarized in the following table, are based on the value of the *return_code* parameter.

return_code	**New state**
CM_OK	RESET
CM_PROGRAM_ERROR_PURGING	RECEIVE
CM_SVC_ERROR_PURGING	RECEIVE
CM_CONVERSATION_TYPE_MISMATCH	RESET
CM_PIP_NOT_SPECIFIED_CORRECTLY	RESET
CM_SECURITY_NOT_VALID	RESET
CM_SYNC_LEVEL_NOT_SUPPORTED_PGM	RESET
CM_TPN_NOT_RECOGNIZED	RESET
CM_TP_NOT_AVAILABLE_NO_RETRY	RESET
CM_TP_NOT_AVAILABLE_RETRY	RESET
CM_RESOURCE_FAILURE_NO_RETRY	RESET
CM_RESOURCE_FAILURE_RETRY	RESET
CM_DEALLOCATED_ABEND	RESET
CM_DEALLOCATED_ABEND_SVC	RESET

CM_DEALLOCATED_ABEND_TIMER	RESET
All others	No change

Remarks

Before deallocating the conversation, this call performs the equivalent of either the Flush or Confirmed call, depending on the current conversation synchronization level and deallocate type. The deallocate type is set by Set_Deallocate_Type.

The partner program receives the deallocation notification through one of the following parameters:

- status_received is CM_CONFIRM_DEALLOC_RECEIVED
- *return_code* is CM_DEALLOCATED_NORMAL
- *return_code* is CM_DEALLOCATED_ABEND

After this call has successfully executed, the *conversation_ID* is no longer valid.

For a basic conversation, if the conversations deallocate type is set to CM_DEALLOCATE_ABEND and the log data length is greater than zero, the local LU writes the log data (specified by Set_Log_Data) to the local error log and to the partner LU.

After **Deallocate** has been executed, the log data length is set to zero and the log data is set to null.

Delete_CPIC_Side_Information

The **Delete_CPIC_Side_Information** call (function name **xcmdsi**) deletes an entry from the side information table in memory. The side information entry is identified through the symbolic destination name.

Syntax

```
CM_ENTRY Delete_CPIC_Side_Information(
  unsigned char FAR *key_lock,
    unsigned char FAR *sym_dest_name,
    CM_INT32 FAR *return_code
);
```

Parameters

key_lock

Supplied parameter. This parameter is ignored.

sym_dest_name

Supplied parameter. Specifies the symbolic destination name of the entry to be deleted.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the *sym_dest_name* parameter specified a nonexistent side information entry.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The call is not associated with a conversation and can be in any state.

There is no state change.

Remarks

The side information entry is removed immediately from the side information table in memory.

While this call is being executed, any calls issued by other CPI-C applications that set or extract side information are suspended. These calls include the following:

- Set_CPIC_Side_Information
- Extract_CPIC_Side_Information
- Initialize_Conversation

Extract_Conversation_Security_Type

The **Extract_Conversation_Security_Type** call (function name **xcecst**) returns the security type for a specified conversation.

Syntax

```
CM_ENTRY Extract_Conversation_Security_Type(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *conversation_security_type,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

conversation_security_type

Returned parameter. Specifies the information the partner logical unit (LU) requires to validate access to the invoked program. Possible values are:

CM_SECURITY_NONE

The invoked program uses no conversation security.

CM_SECURITY_PROGRAM

The invoked program uses conversation security and thus requires a user identifier and password.

CM_SECURITY_SAME

The invoked program, invoked with a valid user identifier and password, in turn invokes another program (as illustrated in Communication Between TPs). For example, assume that program A invokes program B with a valid user identifier and password, and program B in turn invokes program C. If program B specifies the value CM_SECURITY_SAME, CPI-C sends the LU for program C, the user identifier from program A, and an already-verified indicator. This indicator tells program C not to require the password (if program C is configured to accept an already-verified indicator).

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid, or the address of a variable is invalid.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Extract_Conversation_Security_User_ID

The **Extract_Conversation_Security_User_ID** call (function name **cmecsu**) returns the user identifier being used in a specified conversation.

Syntax

```
CM_ENTRY Extract_Conversation_Security_User_ID(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *security_user_ID,
  CM_INT32 FAR *security_user_ID_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

security_user_ID

Returned parameter. Specifies the user identifier that was used to establish the conversation.

security_user_ID_length

Returned parameter. Specifies the length of *security_user_ID*.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

The *security_user_ID* value is not padded with spaces. It is meaningful only up to *security_user_ID_length*.

Extract_Conversation_State

The **Extract_Conversation_State** call (function name **cmecs**) returns the state of the specified conversation.

Syntax

```
CM_ENTRY Extract_Conversation_State(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *conversation_state,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

conversation_state

Returned parameter. Specifies the conversation state. Possible values are:

- CM_INITIALIZE_STATE
- CM_SEND_STATE
- CM_RECEIVE_STATE
- CM_SEND_PENDING_STATE
- CM_CONFIRM_STATE
- CM_CONFIRM_SEND_STATE
- CM_CONFIRM_DEALLOCATE_STATE

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Extract_Conversation_Type

The **Extract_Conversation_Type** call (function name **cmect**) returns the conversation type—mapped or basic—of the specified conversation.

Syntax

```
CM_ENTRY Extract_Conversation_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *conversation_type,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

conversation_type

Returned parameter. Specifies the conversation type. Possible values are:

- CM_BASIC_CONVERSATION
- CM_MAPPED_CONVERSATION

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Extract_CPIC_Side_Information

The **Extract_CPIC_Side_Information** call (function name **xcmesi**) returns the side information for an entry number or symbolic destination name.

Syntax

```
CM_ENTRY Extract_CPIC_Side_Information(
  CM_INT32 FAR *entry_number,
  unsigned char FAR *sym_dest_name,
  SIDE_INFO FAR *side_info_entry,
  CM_INT32 FAR *side_info_entry_length,
    CM_INT32 FAR *return_code
);
```

Parameters

entry_number

Supplied parameter. Specifies the number (index) of the side information entry to be returned. The first entry is 1.

The program can look up the side information entry by the symbolic destination name instead. To accomplish this, set the entry number to zero.

sym_dest_name

Supplied parameter. Specifies the symbolic destination name to search for.

If *entry_number* is set to a number greater than zero, this parameter is ignored.

side_info_entry

Returned parameter. Specifies the side information entry. For a detailed explanation of the side information entry, see Set_CPIC_Side_Information.

Each field in the side information structure is left-aligned and padded with spaces on the right as necessary.

side_info_entry_length

Supplied parameter. Always 124.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The *entry_number* specified a number larger than the maximum number of entries in the side information table or a number that is less than zero.
- The *sym_dest_name* parameter is invalid and *entry_number* is set to zero.
- The *side_info_entry_length* parameter is not set to 124.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

This call is not associated with a conversation and can be in any state.

There is no state change.

Remarks

The security password is never returned. If the security user identifier in the side information is not set, the security user identifier field is returned as all spaces.

Extract_Mode_Name

The **Extract_Mode_Name** call (function name **cmemn**) returns the mode name and mode name length for a specified conversation.

Syntax

```
CM_ENTRY Extract_Mode_Name(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *mode_name,
  CM_INT32 FAR *mode_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

mode_name

Returned parameter. Specifies the starting address of the mode name.

mode_name_length

Returned parameter. Specifies the length of the mode name.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Extract_Partner_LU_Name

The **Extract_Partner_LU_Name** call (function name **cmepln**) returns the partner LU name and partner LU name length for a specified conversation. This can be an alias name of up to eight bytes or a fully qualified network name of up to 17 bytes.

Syntax

```
CM_ENTRY Extract_Partner_LU_Name(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *partner_LU_name,
  CM_INT32 FAR *partner_LU_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

partner_LU_name

Returned parameter. Specifies the variable containing the partner LU name. (The program must supply a pointer to a suitable variable.)

partner_LU_name_length

Returned parameter. Specifies the length of the partner LU name.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

An invokable CPI-C transaction program (TP) will only receive the fully qualified network name upon successful completion of this function call. An invokable CPI-C TP is unable to retrieve the alias name using this call.

Extract_Sync_Level

The **Extract_Sync_Level** call (function name **cmesl**) returns the synchronization level for a specified conversation.

Syntax

```
CM_ENTRY Extract_Sync_Level(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *sync_level,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

sync_level

Returned parameter. Indicates the synchronization level of the conversation. Possible values are:

CM_NONE

The programs will not perform confirmation processing.

CM_CONFIRM

The programs can perform confirmation processing.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Extract_TP_Name

The **Extract_TP_Name** call (function name **cmetpn**) returns the *TP_name* characteristic.

Syntax

```
CM_ENTRY Extract_TP_Name(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *TP_name,
  CM_INT32 FAR *TP_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

TP_name

Returned parameter. Specifies the variable containing the transaction program (TP) name.

TP_name_length

Returned parameter. Specifies the length of the TP name.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

For an invoking program, the *TP_name* characteristic is the value in the side information referenced in the *sym_dest_name* parameter of the Initialize_Conversation call. For an invokable program, it is the name specified in the conversation startup request (which will have been matched with a name specified locally or in a Specify_Local_TP_Name call), and will therefore be the same as the *TP_name* characteristic of the partner program.

The name returned can be up to 64 bytes in length.

Flush

The **Flush** call (function name **cmflus**) sends the contents of the send buffer of the local logical unit (LU) to the partner LU (and program). If the send buffer is empty, no action takes place.

Syntax

```
CM_ENTRY Flush(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation was not in SEND or SEND_PENDING state when the program issued this call.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in SEND or SEND_PENDING state.

If the call completes successfully, (*return_code* is CM_OK), the conversation is in SEND state.

Other return codes result in no state change.

Remarks

Data processed by Send_Data accumulates in the local LUs send buffer until one of the following happens:

- The local program issues the Flush call or other call that flushes the LUs send buffer. (Some send types, set by Set_Send_Type, include flush functionality.)
- The buffer is full.

The allocation request generated by Allocate and error information generated by Send_Error are also buffered.

Initialize_Conversation

The **Initialize_Conversation** call (function name **cminit**) is issued by the invoking program to obtain an 8-byte conversation identifier and to set the initial values for the conversations characteristics.

Syntax

```
CM_ENTRY Initialize_Conversation(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *sym_dest_name,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Returned parameter. Specifies the identifier for the conversation. It is used by subsequent CPI-C calls.

sym_dest_name

Supplied parameter. Specifies the symbolic destination name—the name associated with a side information entry loaded from the configuration file or defined by Set_CPIC_Side_Information calls.

This parameter is an 8-byte ASCII character string. The allowed characters are as follows:

- Uppercase letters
- Numerals from 0 through 9

This parameter can also be set to eight spaces. In this case, the invoking program must issue the following calls before issuing Allocate:

- Set_Mode_Name
- Set_Partner_LU_Name
- Set_TP_Name

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *sym_dest_name* does not match a symbolic destination name in the side information table and is not a space.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation is in RESET state.

If the *return_code* is CM_OK, the conversation changes to INITIALIZE state. For other return codes, the conversation state remains unchanged.

Remarks

The initial values are CPI-C defaults or are derived from side information associated with the symbolic destination name. For more information about initial values and side information, see Initial Conversation Characteristics and Side Information for CPI-C Programs.

Initial values can be changed by the **Set_** calls.

If the side information contains an invalid value or a **Set_** call sets a conversation characteristic to an invalid value, the error is returned on the **Allocate** call.

If a CPI-C application attempts to invoke more than one concurrent conversation, only a single local APPC logical unit (LU) is used by all conversations. This prevents concurrent conversations across two or more dependent LU 6.2 LUs, causing subsequent Initialize_Conversation (CMALLC) calls to wait for the first conversation to be deallocated.

If the CPI-C application needs to invoke more than one concurrent conversation, independent LU 6.2 must be used between Host Integration Server and the remote system.

Upon successful execution of this call, CPI-C generates a conversation identifier. This identifier is a required parameter for all other CPI-C calls issued for this conversation by the invoking program.

Under normal circumstances, a CPI-C application cannot invoke two concurrent conversations using two different local APPC LUs. A registry key is available that when set forces CPI-C to issue a new TP_STARTED verb on every Initialize_Conversation (cminit) call. This is necessary to force APPC resource location for each call. The registry key that must be defined to force this behavior is the following:

\HKLM\CurrentControlSet\Services\SnaBase\Parameters\Client\GETNEWTPID

Prepare_To_Receive

The **Prepare_To_Receive** call (function name **cmptr**) changes the state of the conversation for the local program from SEND to RECEIVE.

Syntax

```
CM_ENTRY Prepare_To_Receive(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The conversation state is not SEND or SEND_PENDING.
- For a basic conversation, the conversation is in SEND state. However, the program did not finish sending a logical record.

These return codes can occur if the conversations prepare-to-receive type is set to CM_PREP_TO_RECEIVE_CONFIRM or if the prepare-to-receive type is set to CM_PREP_TO_RECEIVE_SYNC_LEVEL and the conversations synchronization level is set to CM_CONFIRM.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner logical unit (LU) or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner

program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

CM_SYNC_LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued Send_Error. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameter set to CM_DEALLOCATE_ABEND, or the remote LU did so because of a remote program abnormal-ending condition. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.
- The remote TP terminated normally but did not deallocate the conversation before terminating. Node services at the remote LU deallocated the conversation on behalf of the remote TP.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.

- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued Send_Error with the type parameter set to SVC. Data sent to the partner program may have been purged.

State Changes

The conversation can be in SEND or SEND_PENDING state.

State changes, summarized in the following table, are based on the value of the *return_code* parameter.

return_code	**New state**
CM_OK	RECEIVE
CM_PROGRAM_ERROR_PURGING	RECEIVE
CM_SVC_ERROR_PURGING	RECEIVE
CM_CONVERSATION_TYPE_MISMATCH	RESET
CM_PIP_NOT_SPECIFIED_CORRECTLY	RESET
CM_SECURITY_NOT_VALID	RESET
CM_SYNC_LEVEL_NOT_SUPPORTED_PGM	RESET
CM_TPN_NOT_RECOGNIZED	RESET
CM_TP_NOT_AVAILABLE_NO_RETRY	RESET
CM_TP_NOT_AVAILABLE_RETRY	RESET
CM_DEALLOCATED_ABEND	RESET
CM_RESOURCE_FAILURE_NO_RETRY	RESET
CM_RESOURCE_FAILURE_RETRY	RESET
CM_DEALLOCATED_ABEND_SVC	RESET
CM_DEALLOCATED_ABEND_TIMER	RESET
All others	No change

Before changing the conversation state, this call performs the equivalent of one of the following:

- The Flush call, sending the contents of the local LUs send buffer to the partner LU and program, if either of the following conditions is true:

    - The conversations prepare-to-receive type is set to CM_PREP_TO_RECEIVE_FLUSH.
    - The conversations prepare-to-receive type is set to CM_PREP_TO_RECEIVE_SYNC_LEVEL and the conversations synchronization level is set to CM_NONE.
- The Confirm call, sending the contents of the local LUs send buffer and a confirmation request to the partner program, if either of the following conditions is true:
    - The conversations prepare-to-receive type is set to CM_PREP_TO_RECEIVE_CONFIRM.
    - The conversations prepare-to-receive type is set to CM_PREP_TO_RECEIVE_SYNC_LEVEL and the conversations synchronization level is set to CM_CONFIRM.

The prepare-to-receive type is set by Set_Prepare_To_Receive_Type; the synchronization level is set by Set_Sync_Level.

The conversation cannot change to SEND or SEND_PENDING for the partner program until the partner program receives one of the following values through the *status_received* parameter of the Receive call:

- CM_SEND_RECEIVED
- CM_CONFIRM_SEND_RECEIVED and replies with the Confirmed or Send_Error call

Remarks

After this call has successfully executed, the local program can receive data.

Receive

The **Receive** call (function name **cmrcv**) receives any data currently available from the partner program. For Microsoft® Windows Server™ 2003 and Windows® 2000, run a background thread for all CPI-C communications and preserve the foreground thread for user interface only.

Syntax

```
CM_ENTRY Receive(
unsigned char FAR *conversation_ID,
unsigned char FAR *buffer,
CM_INT32 FAR *requested_length,
CM_INT32 FAR *data_received,
CM_INT32 FAR *received_length,
CM_INT32 FAR *status_received,
CM_INT32 FAR *request_to_send_received,
CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

buffer

Returned parameter. Specifies the address of the buffer to contain the data received by the local program.

The buffer contains data if the following conditions are true:

- The *data_received* parameter is set to a value other than CM_NO_DATA_RECEIVED.
- The *return_code* parameter is set to CM_OK or to CM_DEALLOCATED_NORMAL.

requested_length

Supplied parameter. Indicates the maximum number of bytes of data the local program is to receive. The range is from 0 through 32767.

data_received

Returned parameter. Indicates whether the program received data. Possible values are listed following the Parameters section.

received_length

Returned parameter. Indicates the number of bytes of data the local program received on this **Receive** call. If *return_code* or *data_received* indicates that the program received no data, this number is not relevant.

status_received

Returned parameter. Indicates changes in the status of the conversation. The following are possible values. These codes are not relevant unless *return_code* is set to CM_OK. Possible values are listed following the Parameters section.

request_to_send_received

Returned parameter. Specifies the request-to-send-received indicator. Values are listed following the Parameters section.

return_code

The code returned from this call. Values are listed following the Parameters section.

Values returned in the data_received parameter

These codes are not relevant unless *return_code* is set to CM_OK or CM_DEALLOCATED_NORMAL.

CM_DATA_RECEIVED

Can be returned for a basic conversation if the conversations fill characteristic is set to CM_FILL_BUFFER, indicating that the program is receiving data independent of its logical format. The local program received data until *requested_length* or end of data was reached.

The end of the data is indicated by either a change to another conversation state, based on the *return_code*, *status_received*, and *data_received* parameters, or an error condition. If the conversations receive type is set to CM_RECEIVE_IMMEDIATE, the data received can be less than *requested_length* if a smaller amount of data has arrived from the partner program.

CM_COMPLETE_DATA_RECEIVED

In a mapped conversation, indicates that the local program has received a complete data record or the last part of a data record.

In a basic conversation with the fill characteristic set to CM_FILL_LL, this value indicates that the local program has received a complete logical record or the end of a logical record.

CM_INCOMPLETE_DATA_RECEIVED

In a mapped conversation, indicates that the local program has received an incomplete data record. The *requested_length* parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first **Receive** to read the record). The amount of data received is equal to the *requested_length* parameter.

In a basic conversation with the fill characteristic set to CM_FILL_LL, this value indicates that the local program has received an incomplete logical record. The amount of data received is equal to the *requested_length* parameter. (If the received data was truncated, the length of the data will be less than *requested_length*.)

Upon receiving this value, the local program normally reissues **Receive** to receive the next part of the record.

CM_NO_DATA_RECEIVED

The program did not receive data.

Note that if the *return_code* parameter is set to CM_OK, status information may be available through the *status_received* parameter.

Values returned in the status_received parameter
CM_NO_STATUS_RECEIVED

No conversation status change was received on this call.

CM_SEND_RECEIVED

Indicates, for the partner program, that the conversation has entered RECEIVE state. For the local program, the conversation is now in SEND state if no data was received on this call, or SEND_PENDING state if data was received on this call.

Upon receiving this value, the local program normally uses Send_Data to begin sending data.

CM_CONFIRM_DEALLOC_RECEIVED

Indicates that the partner program issued Deallocate with confirmation requested. For the local program, the conversation is now in CONFIRM_DEALLOCATE state.

Upon receiving this value, the local program normally issues the Confirmed call.

CM_CONFIRM_RECEIVED

Indicates that the partner program issued the Confirm call. For the local program, the conversation is in CONFIRM state.

Upon receiving this value, the local program normally issues the **Confirmed** call.

CM_CONFIRM_SEND_RECEIVED

Indicates, for the partner program, that the conversation has entered RECEIVE state and a request for confirmation has been received by the local program. For the local program, the conversation is now in CONFIRM_SEND state.

The program normally responds by issuing the **Confirmed** call. Upon successful execution of the **Confirmed** call, the conversation changes to SEND state for the local program.

Values returned in the request_to_send_received parameter
CM_REQ_TO_SEND_RECEIVED

The partner program issued the Request_To_Send call, which requests the local program to change the conversation to RECEIVE state.

CM_REQ_TO_SEND_NOT_RECEIVED

The partner program did not issue the **Request_To_Send** call. This value is not relevant if the *return_code* parameter is set to CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK.

Values returned in the return_code parameter
CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Windows® message and not call **Wait_For_Conversation**.

CM_UNSUCCESSFUL

Primary return code; the receive type is set to CM_RECEIVE_IMMEDIATE and no data is immediately available from the partner program.

CM_DEALLOCATED_NORMAL

Primary return code; the conversation has been deallocated normally. The partner program issued Deallocate with the conversations deallocate type set to CM_DEALLOCATE_FLUSH or CM_DEALLOCATE_SYNC_LEVEL with the synchronization level of the conversation specified as CM_NONE.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *requested_length* is out of range (greater than 32767).

If the program receives this return code, the other returned parameters are not valid.

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The receive type is set to CM_RECEIVE_AND_WAIT and the conversation state is not RECEIVE, SEND, or SEND_PENDING.
- The receive type is set to CM_RECEIVE_IMMEDIATE and the conversation state is not RECEIVE.
- In a basic conversation, the conversation is in SEND state, the receive type is set to CM_RECEIVE_AND_WAIT, and the program did not finish sending a logical record.

If the program receives this return code, the other returned parameters are not valid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner logical unit (LU) or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

CM_SYNC_LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_NO_TRUNC

Primary return code; while in SEND state or in SEND_PENDING state with the error direction set to CM_SEND_ERROR, the partner program issued Send_Error. Data was not truncated.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued **Send_Error**. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameterset to CM_DEALLOCATE_ABEND, or the remote LU did so because of a remote program abnormal-ending condition. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.
- The remote TP terminated normally but did not deallocate the conversation before terminating. Node services at the remote LU deallocated the conversation on behalf of the remote TP.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.
- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued Send_Error with the type parameter set to SVC. Data sent to the partner program may have been purged.

CM_SVC_ERROR_NO_TRUNC

Primary return code; while in SEND state, the partner program or partner LU issued **Send_Error** with the type parameter set to SVC. Data sent to the partner program may have been purged.

CM_PROGRAM_ERROR_TRUNC

Primary return code; in SEND state, before finishing sending a complete logical record, the partner program issued **Send_Error**. The local program may have received the first part of the logical record through a **Receive** call.

CM_SVC_ERROR_TRUNC

Primary return code; while in RECEIVE or CONFIRM state, the partner program or partner LU issued **Send_Error** with the type parameter set to SVC before it finished sending a complete logical record. The local program may have received the first part of the logical record.

State Changes

The conversation can be in RECEIVE, SEND, or SEND_PENDING state.

If *receive_type* is set to CM_RECEIVE_IMMEDIATE, the conversation must be in RECEIVE state.

Issuing **Receive** while the conversation is in SEND or SEND_PENDING state causes the local LU to send the information in its send buffer and a send indicator to the partner program. Based on *data_received* and *status_received* the conversation can change to RECEIVE state for the local program.

The new conversation state is determined by:

- The state the conversation is in when the program issues the call.
- The *return_code* parameter.
- The data_received and status_received parameters.

If no data is currently available and the receive type (set by Set_Receive_Type) is set to CM_RECEIVE_AND_WAIT, the local program waits for data to arrive. If the receive type is set to CM_RECEIVE_IMMEDIATE, the local program does not wait.

The process for receiving data is as follows:

- The local program issues a Receive call until it finishes receiving a complete unit of data. The local program may need to issue Receive several times to receive a complete unit of data. The *data_received* parameter indicates whether the receipt of data is finished.

  The data received can be:

  - One data record transmitted in a mapped conversation.
  - One logical record transmitted in a basic conversation with the conversations fill characteristic set to CM_FILL_LL.

- A buffer of data received independent of its logical-record format in a basic conversation with the fill characteristic set to CM_FILL_BUFFER.

When a complete unit of data has been received, the local program can manipulate it.

- The local program determines the next action to take based on the control information received through *status_received*. The local program may have to reissue **Receive** to receive the control information.

The conversation type is set by Set_Conversation_Type. The fill characteristic is set by Set_Fill.

The following table summarizes the state changes that can occur when **Receive** is issued with the conversation in RECEIVE state and *return_code* is CM_OK.

data_received	***status_received***	**New state**
CM_DATA_RECEIVED	CM_NO_STATUS_RECEIVED	No change
CM_COMPLETE_DATA_ RECEIVED	CM_NO_STATUS_RECEIVED	No change
CM_INCOMPLETE_DATA_ RECEIVED	CM_SEND_RECEIVED	SEND_PENDING
CM_NO_DATA_RECEIVED	CM_SEND_RECEIVED	SEND

If *return_code* is set to CM_UNSUCCESSFUL, meaning that the *receive_type* is set to CM_RECEIVE_IMMEDIATE and no data is available, there is no state change.

The following table summarizes the state changes that can occur when **Receive** is issued with the conversation in SEND state and *return_code* is CM_OK.

data_received	***status_received***	**New state**
CM_DATA_RECEIVED	CM_NO_STATUS_RECEIVED	RECEIVE
CM_COMPLETE_DATA_ RECEIVED	CM_NO_STATUS_RECEIVED	RECEIVE
CM_INCOMPLETE_DATA_ RECEIVED	CM_SEND_RECEIVED	SEND_PENDING
CM_NO_DATA_RECEIVED	CM_SEND_RECEIVED	No change

The following table summarizes the state changes that can occur when **Receive** is issued with the conversation in SEND_PENDING state and *return_code* is CM_OK.

data_received	***status_received***	**New state**
CM_DATA_RECEIVED	CM_NO_STATUS_RECEIVED	RECEIVE
CM_COMPLETE_DATA_ RECEIVED	CM_NO_STATUS_RECEIVED	RECEIVE
CM_INCOMPLETE_DATA_ RECEIVED	CM_SEND_RECEIVED	No change
CM_NO_DATA_RECEIVED	CM_SEND_RECEIVED	SEND

The following topics summarize state changes that can occur when **Receive** is issued in any allowed state.

In This Section

- Confirmation
- Normal Deallocation

- ABEND
- Errors

Confirmation

The following table summarizes state changes that occur under the following conditions:

- The return_code parameter is CM_OK.
- The *data_received* parameter is set to CM_DATA_RECEIVED, CM_COMPLETE_DATA_RECEIVED, or CM_NO_DATA_RECEIVED.
- The *status_received* parameter indicates a change to a CONFIRM state.

status_received	**New state**
CM_CONFIRM_DEALLOC_RECEIVED	CONFIRM_DEALLOCATE
CM_CONFIRM_SEND_RECEIVED	CONFIRM_SEND
CM_CONFIRM_RECEIVED	CONFIRM

Normal Deallocation

If *return_code* is set to CM_DEALLOCATED_NORMAL, the conversation changes to RESET state.

ABEND

The following ABEND conditions, indicated by *return_code*, cause the conversation to change to RESET state:

- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LEVEL_NOT_SUPPORTED_PGM
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY
- CM_TP_NOT_AVAILABLE_RETRY
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_ABEND_SVC
- CM_DEALLOCATED_ABEND_TIMER
- CM_SVC_ERROR_TRUNC
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY

Errors

The following table summarizes state changes that occur when a data transmission error is encountered.

return_code	**Old state**	**New state**
CM_PROGRAM_ERROR_PURGING	RECEIVE	No change
CM_PROGRAM_ERROR_NO_TRUNC	RECEIVE	No change
CM_SVC_ERROR_PURGING	SEND	RECEIVE
CM_SVC_ERROR_NO_TRUNC	SEND_PENDING	RECEIVE

If the partner program truncates a logical record, the local program receives notification of the truncation through *return_code* on the next **Receive** call.

If a program issues **Receive** with *requested_length* set to zero, the call is executed as usual. However, *data_received* and *status_received* are not set on the same **Receive** call. (One exception to this situation is the null record sent over a mapped conversation, described in the next paragraph.)

In a mapped conversation in which data is available from the partner program, *data_received* is set to CM_INCOMPLETE_DATA_RECEIVED. If a null record is available (*send_length* in the Send_Data call issued by the partner program is set to zero), *data_received* is set to CM_COMPLETE_RECORD_RECEIVED with *received_length* set to zero.

In a basic conversation in which data is available and the fill characteristic is set to CM_FILL_LL, *data_received* is set to CM_INCOMPLETE_DATA_RECEIVED. If the fill characteristic is set to CM_FILL_BUFFER, *data_received* is set to CM_DATA_RECEIVED.

The logical unit (LU) does not automatically perform any conversion between EBCDIC and ASCII on the received string of data before putting it in *buffer*. If necessary, the program can use the Common Service Verb (CSV) CONVERT to translate a string from one character set to the other.

Request_To_Send

The **Request_To_Send** call (function name **cmrts**) notifies the partner program that the local program wants to send data.

Syntax

```
CM_ENTRY Request_To_Send(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in the RECEIVE, SEND, SEND_PENDING, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any of the following states: RECEIVE, SEND, SEND_PENDING, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

There is no state change.

In response to this request, the partner program can change the conversation to RECEIVE state by issuing one of the following calls:

- Receive with receive type set to CM_RECEIVE_AND_WAIT
- Prepare_To_Receive
- **Send_Data** with send type set to CM_SEND_AND_PREP_TO_RECEIVE

The partner program can also ignore the request to send.

The conversation state changes to SEND for the local program when the local program receives one of the following values through the *status_received* parameter of a subsequent **Receive** call:

- CM_SEND_RECEIVED
- CM_CONFIRM_SEND_RECEIVED and the local program replies with a Confirmed call

Remarks

The request-to-send notification is received by the partner program through the *request_to_send_received* parameter of the following calls:

- Confirmed
- Receive
- Send_Data
- Send_Error
- Test_Request_To_Send_Received

Request-to-send notification is sent to the partner program immediately. CPI-C does not wait until the send buffer fills up or is flushed. Consequently, the request-to-send notification can arrive out of sequence. For example, if the local program is in SEND state and issues the Prepare_To_Receive call followed by the **Request_To_Send** call, the partner program, in RECEIVE state, can receive the request-to-send notification before it receives the send notification. For this reason, *request_to_send* can be reported to a program through the Receive call.

Upon receiving a request-to-send notification, the partner logical unit (LU) retains the notification until the partner issues a call that returns *request_to_send_received*. The LU keeps only one request-to-send notification per conversation. Thus the local program can issue more **Request_To_Send** calls than are explicitly handled by the partner transaction program (TP).

Send_Data

The **Send_Data** call (function name **cmsend**) puts data in the send buffer of the local logical unit (LU) for transmission to the partner program.

Syntax

```
CM_ENTRY Send_Data(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *buffer,
    CM_INT32 FAR *send_length,
    CM_INT32 FAR *request_to_send_received,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

buffer

Supplied parameter. Specifies the address of the buffer containing the data to be put in the local LUs send buffer.

send_length

Supplied parameter. Specifies the number of bytes of data to be put in the local LUs send buffer. The range is from 0 through 32767.

For a mapped conversation, if *send_length* is set to zero, a null data record is sent to the partner program.

For a basic conversation, if *send_length* is set to zero, no data is sent. The *buffer* parameter is not relevant. However, the other parameters are processed.

request_to_send_received

Returned parameter. It is the request-to-send-received indicator. Possible values are:

CM_REQ_TO_SEND_RECEIVED

The partner program issued the Request_To_Send call, which requests the local program to change the conversation to RECEIVE state.

CM_REQ_TO_SEND_NOT_RECEIVED

The partner program did not issue the **Request_To_Send** call. This value is not relevant if *return_code* is set to CM_PROGRAM_PARAMETER_CHECK or CM_PROGRAM_STATE_CHECK.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *send_length* is out of range (greater than 32767).
- This is a basic conversation and the first two bytes of *buffer* contain an invalid logical record length (0x0000, 0x0001, 0x8000, or 0x8001).

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The conversation state is not SEND or SEND_PENDING.
- The basic conversation is in SEND state and *send_type* is set to CM_SEND_AND_CONFIRM, CM_SEND_AND_DEALLOCATE, or CM_SEND_AND_PREP_TO_RECEIVE. However, the data does not end on a logical record boundary. This condition is allowed only when *deallocate_type* is set to CM_DEALLOCATE_ABEND and the *send_type* is set to CM_SEND_AND_DEALLOCATE.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner LU or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

CM_SYNC_LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued Send_Error. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameter set to CM_DEALLOCATE_ABEND, or the remote LU did so because of a remote program abnormal-ending condition. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.
- The remote TP terminated normally but did not deallocate the conversation before terminating. Node services at the remote LU deallocated the conversation on behalf of the remote TP.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.
- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued Send_Error with the type parameter set to SVC. Data sent to the partner program may have been purged.

State Changes

The conversation must be in SEND or SEND_PENDING state when the program issues this call.

The following table summarizes state changes that are possible when *return_code* is set to CM_OK.

send_type	**Old state**	**New state**
CM_BUFFER_DATA	SEND	No change
CM_BUFFER_DATA	SEND_PENDING	SEND
CM_SEND_AND_FLUSH	SEND	No change
CM_SEND_AND_FLUSH	SEND_PENDING	SEND

CM_SEND_AND_CONFIRM	SEND	No change
CM_SEND_AND_CONFIRM	SEND_PENDING	SEND
CM_SEND_AND_PREP_TO_ RECEIVE	Not available	RECEIVE
CM_SEND_AND_DEALLOCATE	Not available	RESET

For a *return_code* value of CM_PROGRAM_ERROR_PURGING or CM_SVC_ERROR_PURGING, the conversation changes to RECEIVE state. For other non-CM_OK values, the conversation changes to RESET state.

Remarks

The data collected in the local LUs send buffer is transmitted to the partner LU and partner program when one of the following occurs:

- The send buffer fills up.
- The local program issues a Flush, Confirm, or Deallocate call or other call that flushes the LUs send buffer. (Some send types, set by Set_Send_Type, include flush functionality.)

The data to be sent can be either:

- A complete data record on a mapped conversation. A complete data record is a string of the length specified by the *send_length* parameter.
- A complete logical record or portion thereof on a basic conversation. A complete logical record is determined by the LL value. (One logical record can end and a new one begin in the middle of the string of data to be sent.)

The LU does not automatically perform any conversion between ASCII and EBCDIC on the string of data to be sent. If necessary, the program can use the Common Service Verb (CSV) CONVERT to translate a string from one character set to the other.

Send_Error

The **Send_Error** call (function name **cmserr**) notifies the partner program that the local program has encountered an application-level error.

Syntax

```
CM_ENTRY Send_Error(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *request_to_send_received,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

request_to_send_received

Returned parameter. Specifies the request-to-send-received indicator. Possible values are:

CM_REQ_TO_SEND_RECEIVED

The partner program issued Request_To_Send, which requests the local program to change the conversation to RECEIVE state.

CM_REQ_TO_SEND_NOT_RECEIVED

The partner program did not issue **Request_To_Send**. This value is not relevant if *return_code* is set to CM_PROGRAM_PARAMETER_CHECK or CM_STATE_CHECK.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

The value of *return_code* varies depending on the conversation state when the call is issued.

SEND State

If the program issues the call with the conversation in SEND state, the following return codes are possible:

CM_OK

Primary return code; the call executed successfully.

CM_OPERATION_NOT_ACCEPTED

Primary return code; a previous operation on this conversation is incomplete.

CM_OPERATION_INCOMPLETE

Primary return code; the operation has not completed (processing mode is nonblocking only) and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a Microsoft® Windows® message and not call **Wait_For_Conversation**.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

CM_CONVERSATION_TYPE_MISMATCH

Primary return code; the partner logical unit (LU) or program does not support the conversation type (basic or mapped) specified in the allocation request.

CM_PIP_NOT_SPECIFIED_CORRECTLY

Primary return code; the allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

CM_SECURITY_NOT_VALID

Primary return code; the user identifier or password specified in the allocation request was not accepted by the partner LU.

CM_SYNC LEVEL_NOT_SUPPORTED_PGM

Primary return code; the partner program does not support the synchronization level specified in the allocation request.

CM_TPN_NOT_RECOGNIZED

Primary return code; the partner LU does not recognize the program name specified in the allocation request.

CM_TP_NOT_AVAILABLE_NO_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

CM_TP_NOT_AVAILABLE_RETRY

Primary return code; the partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

CM_PROGRAM_ERROR_PURGING

Primary return code; one of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued **Send_Error**. Data sent but not yet received is purged.
- While in SEND_PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

CM_RESOURCE_FAILURE_NO_RETRY

Primary return code; one of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

CM_RESOURCE_FAILURE_RETRY

Primary return code; the conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

CM_DEALLOCATED_ABEND

Primary return code; the conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameterset to CM_DEALLOCATE_ABEND, or the remote LU has done so because of a remote program abnormal-ending condition. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not yet received by the remote program is purged.
- The remote TP terminated normally but did not deallocate the conversation before terminating. Node services at the remote LU deallocated the conversation on behalf of the remote TP.

CM_DEALLOCATED_ABEND_SVC

Primary return code; the conversation has been deallocated for one of the following reasons:

- The partner program issued **Deallocate** with the type parameter set to ABEND_SVC.

- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_DEALLOCATED_ABEND_TIMER

Primary return code; the conversation has been deallocated because the partner program issued **Deallocate** with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

CM_SVC_ERROR_PURGING

Primary return code; while in SEND state, the partner program or partner LU issued **Send_Error** with the type parameter set to SVC. Data sent to the partner program may have been purged.

RECEIVE State

If the call is issued in RECEIVE state, the following return codes are possible:

CM_OK

Primary return code; because incoming information is purged when the **Send_Error** call is issued in RECEIVE state, CM_OK is generated instead of the following:

- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING
- CM_SVC_ERROR_NO_TRUNC
- CM_SVC_ERROR_PURGING
- CM_PROGRAM_ERROR_TRUNC
- CM_SVC_ERROR_TRUNC (basic conversation only)
- CM_PRODUCT_SPECIFIC_ERROR
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY

For an explanation of these return codes, see CPI-C Common Return Codes.

CM_DEALLOCATED_NORMAL

Primary return code; because incoming information is purged when **Send_Error** is issued in RECEIVE state, CM_DEALLOCATED_NORMAL is generated instead of the following:

- CM_CONVERSATION_TYPE_MISMATCH
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_SECURITY_NOT_VALID
- CM_SYNC_LEVEL_NOT_SUPPORTED_PGM
- CM_TPN_NOT_RECOGNIZED
- CM_TP_NOT_AVAILABLE_NO_RETRY

- CM_TP_NOT_AVAILABLE_RETRY
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_ABEND_SVC
- CM_DEALLOCATED_ABEND_TIMER

SEND_PENDING State

If the call is issued in SEND_PENDING state, the following return codes are possible:

- CM_OK (Primary return code; the call executed successfully.)
- CM_PRODUCT_SPECIFIC_ERROR
- CM_PROGRAM_ERROR_PURGING
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY
- CM_DEALLOCATED_ABEND
- CM_DEALLOCATED_ABEND_SVC
- CM_DEALLOCATED_ABEND_TIMER
- CM_SVC_ERROR_PURGING

For an explanation of these return codes, see CPI-C Common Return Codes.

CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE State

If the call is issued in CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state, the following return codes are possible:

- CM_OK (Primary return code; the call executed successfully.)
- CM_PRODUCT_SPECIFIC_ERROR
- CM_RESOURCE_FAILURE_NO_RETRY
- CM_RESOURCE_FAILURE_RETRY

For an explanation of these return codes, see CPI-C Common Return Codes.

Other States

Issuing **Send_Error** with the conversation in RESET or INITIALIZE state is illegal. The following return codes are possible:

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation state is not SEND, RECEIVE, CONFIRM, CONFIRM_SEND, CONFIRM_DEALLOCATE, or SEND_PENDING.

State Changes

The conversation can be in any state except INITIALIZE or RESET.

State changes, summarized in the following table, are based on the value of the *return_code* parameter.

return_code	**New state**
CM_OK	SEND
CM_CONVERSATION_TYPE_MISMATCH	RESET
CM_PIP_NOT_SPECIFIED_CORRECTLY	RESET
CM_SECURITY_NOT_VALID	RESET
CM_SYNC_LEVEL_NOT_SUPPORTED_PGM	RESET
CM_TPN_NOT_RECOGNIZED	RESET
CM_TP_NOT_AVAILABLE_NO_RETRY	RESET
CM_TP_NOT_AVAILABLE_RETRY	RESET
CM_RESOURCE_FAILURE_RETRY	RESET
CM_RESOURCE_FAILURE_NO_RETRY	RESET
CM_DEALLOCATED_ABEND	RESET
CM_DEALLOCATED_ABEND_PROG	RESET
CM_DEALLOCATED_ABEND_SVC	RESET
CM_DEALLOCATED_ABEND_TIMER	RESET
CM_DEALLOCATED_NORMAL	RESET
CM_PROGRAM_ERROR_PURGING	RECEIVE
CM_SVC_ERROR_PURGING	RECEIVE
All others	No change

Upon successful execution of this call, the conversation is in SEND state for the local program and in RECEIVE state for the partner program.

In a basic conversation, the local program can use Set_Log_Data to specify that error log data be sent to the partner LU and added to the local error log. If the conversations log data length characteristic is greater than zero, the LU formats the data and stores it in the send buffer.

After **Send_Error** is completed, the log data length is set to zero and the log data to null.

If the conversation is in RECEIVE state when the program issues **Send_Error**, incoming data is purged by CPI-C. This data includes:

- Data sent by Send_Data.
- Confirmation requests.
- Deallocation requests if the conversations deallocate type is set to CM_DEALLOCATE_CONFIRM or to

CM_DEALLOCATE_SYNC_LEVEL with the synchronization level set to CM_CONFIRM.

CPI-C does notpurge an incoming request-to-send indicator.

If the conversation is in SEND_PENDING state, the local program can issue Set_Error_Direction to specify whether the error being reported resulted from the received data or from the processing of the local program after successfully receiving the data.

Remarks

The local program can use **Send_Error** for such purposes as informing the partner program of an error encountered in received data, rejecting a confirmation request, or truncating an incomplete logical record it is sending.

Send_Error flushes the local LUs send buffer and sends the partner program the contents of the send buffer followed by the error notification.

The error notification is sent to the partner as one of the following *return_code* values:

- CM_PROGRAM_ERROR_TRUNC
- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING

Set_Conversation_Security_Password

The **Set_Conversation_Security_Password** call (function name **cmscsp**) is issued by the invoking program to specify the password required to gain access to the invoked program.

Syntax

```
CM_ENTRY Set_Conversation_Security_Password(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *security_password,
  CM_INT32 FAR *security_password_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

security_password

Supplied parameter. Specifies the password required to gain access to the partner program. This parameter is a character string of up to eight ASCII characters and is case-sensitive. It must match the password for the user identifier configured for the partner program.

The allowed characters are:

- Uppercase and lowercase letters.
- Numerals from 0 through 9.
- Special characters, except the space.

If the CPI-C automatic logon feature is to be used, this parameter must be set to the MS$SAME string. For details, see the Remarks section later in this topic.

security_password_length

Supplied parameter. Specifies the length of *security_password*. The range is from 0 through 8.

If the CPI-C automatic logon feature is to be used, this parameter must be set to 7. For details, see the Remarks section later in this topic.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *security_password_length* is out of range.

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The conversation is not in INITIALIZE state.
- The conversations security type is not set to CM_SECURITY_PROGRAM.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

This call has an effect on the conversation only if the conversation security type is CM_SECURITY_PROGRAM or CM_SECURITY_SAME. It overrides the initial password from the side information specified by Initialize_Conversation. This call cannot be issued after Allocate has been issued.

An invalid password is not detected until the allocation request, generated by **Allocate**, is sent to the partner logical unit (LU). The error is returned to the invoking program on a subsequent call.

Automatic logon for CPI-C applications is supported by Host Integration Server 2009. This feature requires specific configuration by the network administrator: The CPI-C application must be invoked on the LAN side from a client of SNA Server. The client must be logged into a Microsoft Windows Server 2003 or Windows 2000 domain, but can be any platform that supports Host Integration Server CPI-C APIs.

The client application is coded to use program level security, with a special hard-coded CPI-C user name MS$SAME and password MS$SAME. When this session allocation flows from client to SNA Server, the SNA Server looks up the host account and password corresponding to the Windows Server 2003 or Windows 2000 account under which the client is logged in, and substitutes the host account information into the APPC attach message it sends to the host.

Set_Conversation_Security_Type

The **Set_Conversation_Security_Type** call (function name **cmscst**) is issued by the invoking program to specify the information the partner logical unit (LU) requires to validate access to the invoked program.

Syntax

```
CM_ENTRY Set_Conversation_Security_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *conversation_security_type,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

conversation_security_type

Supplied parameter. Specifies the information the partner LU requires to validate access to the invoked program. Based on the conversation security established for the invoked program during configuration, use one of the following values:

CM_SECURITY_NONE

To indicate that the invoked program uses no conversation security.

CM_SECURITY_PROGRAM

To indicate that the invoked program uses conversation security and thus requires a user identifier and password.

CM_SECURITY_SAME

To indicate that the user ID is sent on the allocate request to node services in the partner LU. This setting is also used to specify that the invoked program, invoked with a valid user identifier and password, in turn invokes another program (as illustrated in Communication Between TPs). For example, assume that program A invokes program B with a valid user identifier and password, and program B in turn invokes program C. If program B specifies the value CM_SECURITY_SAME, CPI-C will send the LU for program C, the user identifier from program A, and an already-verified indicator. This indicator tells program C not to require the password (if program C is configured to accept an already-verified indicator).

When CM_SECURITY_SAME is used, your application must always call Set_Conversation_Security_User_ID and Set_Conversation_Security_Password to provide values for the *security_user_ID* and *security_password* parameters. Depending on the properties negotiated between SNA Server and the peer LU, the **Allocate** function will send one of 3 kinds of Attach (FMH-5) messages, in this order of precedence:

1. If the LUs have negotiated already verified security, the Attach sent by SNA Server will not include the contents of the security_password parameter field specified by Set_Conversation_Security_Password.

2. If the LUs have negotiated persistent verification security, the Attach sent by SNA Server will include the security_password parameter specified by Set_Conversation_Security_Password, but only when the Attach is the first for the specified security_user_ID parameter set by Set_Conversation_Security_User_ID since the start of the LU-LU session, and will omit the security_password parameter on all subsequent Attaches (issued by your application or any other application using this LU-LU-mode triplet).

3. Your application cannot tell which mode of security has been negotiated between the LUs, nor can it tell whether the **Allocate** function it is issuing is the first for that LU-LU-mode triplet. So your application must always call Set_Conversation_Security_User_ID and Set_Conversation_Security_Password to set the security_user_ID and security_password parameters when conversation_security_type is set to CM_SECURITY_SAME.

For more information on persistent verification and already verified security, see the SNA Formats Guide, section "FM Header 5: Attach (LU 6.2)."

If the CPI-C automatic logon feature is to be used, this parameter must be set to CM_SECURITY_PROGRAM. For details, see the Remarks section later in this topic.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* or *conversation_security_type* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

This call overrides the initial security type from the side information specified by Initialize_Conversation. This call cannot be issued after Allocate has been issued.

If the conversation security type is set to CM_SECURITY_NONE, the user identifier and password are ignored when the conversation is allocated.

A conversation security type of CM_SECURITY_SAME is intended for use between nodes which have the same set of user IDs and which accept user validation performed on one node as validating the user for all nodes. A password is not used in this case except for the initial validation of the user ID.

Automatic logon for CPI-C applications is supported by Host Integration Server 2009. This feature requires specific configuration by the network administrator. The CPI-C application must be invoked on the LAN side from a client of SNA Server. The client must be logged into a Microsoft Windows Server 2003 or Windows 2000 domain, but can be any platform that supports SNA Server CPI-C APIs.

The client application is coded to use program level security, with a special hard-coded CPI-C user name MS$SAME and password MS$SAME. When this session allocation flows from client to SNA Server, the SNA Server looks up the host account and password corresponding to the Windows Server 2003 or Windows 2000 account under which the client is logged on, and substitutes the host account information into the APPC attach message it sends to the host.

Set_Conversation_Security_User_ID

The **Set_Conversation_Security_User_ID** call (function name **cmscsu**) is issued by the invoking program to specify the user identifier required to gain access to the invoked program.

Syntax

```
CM_ENTRY Set_Conversation_Security_User_ID(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *security_user_ID,
  CM_INT32 FAR *security_user_ID_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

security_user_ID

Supplied parameter. Specifies the user identifier required to gain access to the partner program. This parameter is a character string of up to eight ASCII characters and is case-sensitive.

The allowed characters are:

- Uppercase and lowercase letters.
- Numerals from 0 through 9.
- Special characters, except the space.

If the CPI-C automatic logon feature is to be used, this parameter must be set to the MS$SAME string. For details, see the Remarks section later in this topic.

security_user_ID_length

Supplied parameter. Specifies the length of *security_user_ID*. The range is from 0 through 8.

If the CPI-C automatic logon feature is to be used, this parameter must be set to 7. For details, see the Remarks section later in this topic.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *security_user_ID_length* is out of range.

CM_PROGRAM_STATE_CHECK

Primary return code; one of the following occurred:

- The conversation is not in INITIALIZE state.

- The conversations security type is not set to CM_SECURITY_PROGRAM.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

This call has an effect on the conversation only if the conversation security type is CM_SECURITY_PROGRAM or CM_SECURITY_SAME. It overrides the initial user identifier from the side information specified by Initialize_Conversation. This call cannot be issued after Allocate has been issued.

An invalid user identifier is not detected until the allocation request, generated by **Allocate**, is sent to the partner logical unit (LU). The error is returned to the invoking program on a subsequent call.

Automatic logon for CPI-C applications is supported by Host Integration Server 2009. This feature requires specific configuration by the network administrator. The CPI-C application must be invoked on the LAN side from a client of SNA Server. The client must be logged into a Microsoft Windows Server 2003 or Windows 2000 domain, but can be any platform that supports SNA Server CPI-C APIs.

The client application is coded to use program level security, with a special hard-coded CPI-C user name MS$SAME and password MS$SAME. When this session allocation flows from client to SNA Server, the SNA Server looks up the host account and password corresponding to the Windows Server 2003 or Windows 2000 account under which the client is logged on, and substitutes the host account information into the APPC attach message it sends to the host.

Set_Conversation_Type

The **Set_Conversation_Type** call (function name **cmsct**) is issued by the invoking program to define a conversation as being mapped or basic. This call overrides the default conversation type established by Initialize_Conversation. The default conversation type is CM_MAPPED_CONVERSATION. This call cannot be issued after Allocate has been issued.

Syntax

```
CM_ENTRY Set_Conversation_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *conversation_type,
    CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

conversation_type

Supplied parameter. Specifies the type of conversation to be allocated by **Allocate**. Possible values are:

- CM_BASIC_CONVERSATION
- CM_MAPPED_CONVERSATION

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* or *conversation_type* is invalid.
- The *conversation_type* parameter specifies a mapped conversation, but the fill characteristic is set to CM_FILL_BUFFER, which is incompatible with mapped conversations. Before changing the conversation type to mapped, you must issue the Set_Fill call to change the fill type to CM_FILL_LL.
- The *conversation_type* parameter specifies a mapped conversation. However, a previous Set_Log_Data call, allowed only in basic conversations, is still in effect.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Set_CPIC_Side_Information

The **Set_CPIC_Side_Information** call (function name **xcmssi**) adds or replaces a side information entry in memory. A CPI-C side information entry associates a set of conversation characteristics with a symbolic definition name. This call overrides entries having the same symbolic destination name.

Syntax

```
CM_ENTRY Set_CPIC_Side_Information(
  unsigned char FAR *key_lock,
  SIDE_INFO FAR *side_info_entry,
  CM_INT32 FAR *side_info_entry_length,
  CM_INT32 FAR *return_code
);
```

Parameters

key_lock

Supplied parameter. This parameter is ignored.

side_info_entry

Supplied parameter. Specifies the contents of a side information entry. The following table describes the *side_info_entry* structure, which defines the format of the side information entry.

Offset	Description	Type	Length
0	*sym_dest_name*	unsigned char	8 bytes
8	*partner_LU_name*	unsigned char	17 bytes
25	*reserved*	unsigned char	3 bytes
28	*TP_name_type*	signed long int	32 bits
32	*TP_name*	unsigned char	64 bytes
96	*mode_name*	unsigned char	8 bytes
104	*conversation_ security_type*	signed long int	32 bits
108	*security_user_ID*	unsigned char	8 bytes
116	*security_password*	unsigned char	8 bytes

The allowed characters for *sym_dest_name* are the uppercase letters (A through Z) and the numerals from 0 through 9.

Set_CPIC_Side_Information is the only CPI-C call that lets you specify an SNA service transaction program (TP) as the partner program. The SNA convention for naming a service TP is up to four characters. The first character is a hexadecimal byte between 0x00 and 0x3F. The remaining characters are translated from ASCII to EBCDIC.

For the allowed characters for the other fields, see the description of the corresponding **Set_** call. For example, for the *mode_name* field, see the description of the Set_Mode_Name call.

Each field in the structure must be left-aligned. Pad fields on the right with spaces as necessary.

side_info_entry_length

Supplied parameter. Specifies the length of *side_info_entry*. It is always 124.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- A value specified in the *side_info_entry* structure is invalid.
- The left character of the *side_info_entry* contains a space.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state.

There is no state change.

Remarks

Invalid string parameters in the side information (for example, specifying a nonexistent partner logical unit (LU)) are not detected until Allocate is issued. The error is returned on a call following **Allocate**.

Set_Deallocate_Type

The **Set_Deallocate_Type** call (function name **cmsdt**) specifies how the conversation is to be deallocated.

Syntax

```
CM_ENTRY Set_Deallocate_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *deallocate_type,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

deallocate_type

Supplied parameter. Specifies how to perform the deallocation. Possible values are:

CM_DEALLOCATE_ABEND

Indicates that the conversation is to be deallocated abnormally and unconditionally. A program should specify CM_DEALLOCATE_ABEND when it encounters an error preventing the successful completion of a transaction.

If the conversation is in SEND state, CPI-C sends the contents of the send buffer of the local logical unit (LU) to the partner program before deallocating the conversation. If the conversation is in RECEIVE state, incoming data can be purged. For a basic conversation in SEND state, logical record truncation can occur.

CM_DEALLOCATE_CONFIRM

Used to send the partner program the contents of the local LUs send buffer and a request to confirm the deallocation.

This request for deallocation confirmation is sent by Deallocate or by Send_Data with the send type set to CM_SEND_AND_DEALLOCATE. The conversation is deallocated normally when the partner program issues Confirmed, responding to the confirmation request.

CM_DEALLOCATE_FLUSH

Used to send the contents of the local LUs send buffer to the partner program before deallocating the conversation normally.

CM_DEALLOCATE_SYNC_LEVEL

Uses the conversations synchronization level to determine how to deallocate the conversation. A default synchronization level is established by Initialize_Conversation and can be overridden by Set_Sync_Level.

If the synchronization level of the conversation is CM_NONE, the default, the contents of the local LUs send buffer are sent to the partner program and the conversation is deallocated normally.

If the synchronization level of the conversation is CM_CONFIRM, the contents of the local LUs send buffer and a request to confirm the deallocation are sent to the partner program. This request for deallocation confirmation is sent by Deallocate or by Send_Data with the send type set to CM_SEND_AND_DEALLOCATE. The conversation is deallocated normally when the partner program issues the Confirmed call, responding to the confirmation request.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* or *deallocate_type* is invalid.
- The *deallocate_type* parameter specifies CM_DEALLOCATE_CONFIRM, but the conversations synchronization level is set to CM_NONE.
- The address of a variable is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

This call overrides the default deallocate type established by Initialize_Conversation or Accept_Conversation. The default deallocate type is CM_DEALLOCATE_SYNC_LEVEL.

The deallocation instructions specified by this call take effect when Deallocate is issued or when the send type is set to CM_SEND_AND_DEALLOCATE and Send_Data is issued.

You can set *deallocate_type* to CM_FLUSH if the synchronization level of the conversation is set to CM_NONE or CM_CONFIRM.

The value CM_DEALLOCATE_FLUSH is functionally the same as CM_DEALLOCATE_SYNC_LEVEL with the conversations synchronization level set to CM_NONE.

The value CM_DEALLOCATE_CONFIRM is functionally the same as CM_DEALLOCATE_SYNC_LEVEL with the conversations synchronization level set to CM_CONFIRM.

Set_Error_Direction

The **Set_Error_Direction** call (function name **cmsed**) specifies whether a program detected an error while receiving data or while preparing to send data.

Syntax

```
CM_ENTRY Set_Error_Direction(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *error_direction,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

error_direction

Supplied parameter. Specifies the direction in which data was flowing when the program encountered an error. Possible values are:

- CM_RECEIVE_ERROR

  An error occurred in the data received from the partner program.

- CM_SEND_ERROR

  An error occurred while the local program prepared to send data to the partner program.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* or *error_direction* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

This call overrides the default error direction established by Initialize_Conversation or Accept_Conversation. The default error direction is CM_RECEIVE_ERROR.

Error direction is relevant only when a program issues Send_Error in SEND_PENDING state, immediately after issuing Receiveand receiving data (*data_received* is a value other than CM_NO_DATA_RECEIVED) and a send indicator (*status_received* is CM_SEND_RECEIVED).

When the conversation is in SEND_PENDING state, the program issues **Send_Error** if it detects errors in the received data or if an error occurred while the local program prepared to send data. The program must supply the error direction information

using **Set_Error_Direction** before issuing **Send_Error** because the logical unit (LU) cannot tell which kind of error occurred (receive or send). The new error direction remains in effect until a subsequent **Set_Error_Direction** changes it.

When **Send_Error** is issued, the partner program receives one of the following return codes:

- CM_PROGRAM_ERROR_PURGING if *error_direction* is set to CM_RECEIVE_ERROR
- CM_PROGRAM_ERROR_NO_TRUNC if *error_direction* is set to CM_SEND_ERROR

Set_Fill

The **Set_Fill** call (function name **cmsf**) specifies whether programs will receive data in the form of logical records or as a specified length of data. This call is allowed only in basic conversations.

Syntax

```
CM_ENTRY Set_Fill(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *fill,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

fill

Supplied parameter. Specifies the form in which programs will receive data. The following are possible choices:

CM_FILL_BUFFER

The local program receives data until the number of bytes specified by the *requested_length* parameter of the **Receive** call is reached or until the end of the data. Data is received without regard for the logical-record format.

CM_FILL_LL

Data is received in logical-record format. The data received can be a complete logical record, a portion of a logical record equal to the *requested_length* parameter of the Receive call, or the end of a logical record.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* or *fill* is invalid.
- The current conversation is mapped.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

Set_Fill overrides the default *fill* established by Initialize_Conversation or Accept_Conversation. The default *fill* is CM_FILL_LL.

The *fill* value affects all subsequent Receive calls. It can be changed by reissuing the **Set_Fill** call.

Set_Log_Data

The **Set_Log_Data** call (function name **cmsld**) specifies a log message (log data) and its length to be sent to the partner logical unit (LU). This call is allowed only in basic conversations. It overrides the default log data, which is null, and the default log data length, which is zero.

Syntax

```
CM_ENTRY Set_Log_Data(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *log_data,
  CM_INT32 FAR *log_data_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

log_data

Supplied parameter. Specifies the starting address of the data to be sent to the partner LU. It can contain up to 512 ASCII characters. The allowed characters are:

- Uppercase and lowercase letters.
- Numerals from 0 through 9.
- Special characters.
- The space.

log_data_length

Supplied parameter. Specifies the length of the log data. The range is from 0 through 512 bytes.

A length of 0 indicates that there is no log data, and the *log_data* parameter is ignored.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The conversation type is set to mapped.
- The value specified by *log_data_length* is out of range (greater than 512 or less than 0).

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

The log data specified by **Set_Log_Data** is sent to the partner LU when the local program issues one of the following calls:

- Send_Error
- Deallocate with the conversations deallocate type set to CM_DEALLOCATE_ABEND
- Send_Data with the conversations send type set to CM_SEND_AND_DEALLOCATE and the deallocate type set to CM_DEALLOCATE_ABEND

After sending the log data to the partner LU, the local LU resets the log data to null and the log data length to zero.

CPI-C automatically converts the log data from ASCII to other encoding standards, such as EBCDIC, as required.

Set_Mode_Name

The **Set_Mode_Name** call (function name **cmsmn**) is issued by the invoking program to specify the mode name for a conversation. This call overrides the system-defined mode name derived from the side information when the Initialize_Conversation call was issued. This call cannot be issued after Allocatehas been issued. Issuing this call has no effect on the side information itself.

Syntax

```
CM_ENTRY Set_Mode_Name(
  unsigned char FAR *conversation_ID,
  unsigned char FAR *mode_name,
  CM_INT32 FAR *mode_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

mode_name

Supplied parameter. Specifies the starting address of the mode name (the name of a set of networking characteristics defined during configuration). The mode name can contain up to eight ASCII characters. The allowed characters are:

- Uppercase letters.
- Numerals from 0 through 9.

The value of *mode_name* must match the name of a mode associated with the partner logical unit (LU) during configuration. The mode name cannot be SNASVCMG or CPSVCMG.

mode_name_length

Supplied parameter. Specifies the length of the mode name. The range is from 0 through 8 bytes.

If *mode_name_length* is set to zero, **Set_Mode_Name** is ignored.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *mode_name_length* is out of range (greater than 8 or less than 0).

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

Specifying an invalid value for *mode_name* is not detected until Allocate is issued.

Set_Partner_LU_Name

The **Set_Partner_LU_Name** call (function name **cmspln**) is issued by the invoking program to specify the partner logical unit (LU) name. This call overrides the partner LU name derived from the side information when the Initialize_Conversation call was issued. This call cannot be issued after Allocate has been issued. Issuing this call has no effect on the side information itself.

Syntax

```
CM_ENTRY Set_Partner_LU_Name(
  unsigned char FAR *conversation_ID,
    unsigned char FAR *partner_LU_name,
  CM_INT32 FAR *partner_LU_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

partner_LU_name

Supplied parameter. Specifies the starting address of the partner LU name. The mode name can contain up to 17 ASCII characters. The allowed characters are:

- Uppercase letters.
- Numerals from 0 through 9.

The partner LU name can be either:

- An alias consisting of one through eight characters.
- A fully qualified network name consisting of from 2 through 17 characters. A period separates the network identifier (which can be from zero through eight characters) from the network LU name (which can be from one through eight characters). If the network identifier is zero characters long, the period is still required.

The partner LU name must match the name of a partner LU established during configuration.

partner_LU_name_length

Supplied parameter. Specifies the length of the partner LU name. The range is from 1 through 17.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *partner_LU_name_length* is out of range (greater than 17 or less than 1).

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

Specifying an invalid value for *partner_LU_name* is not detected until Allocate is issued.

Set_Prepare_To_Receive_Type

The **Set_Prepare_To_Receive_Type** call (function name **cmsptr**) specifies how the subsequent Prepare_To_Receive calls will be executed. It overrides the default prepare-to-receive processing established by Initialize_Conversation or Accept_Conversation. By default, the prepare-to-receive processing is based on the synchronization level of the conversation.

The prepare-to-receive type affects all subsequent **Prepare_To_Receive** calls. It can be changed by reissuing **Set_Prepare_To_Receive_Type**.

Syntax

```
CM_ENTRY Set_Prepare_To_Receive_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *prepare_to_receive_type,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

prepare_to_receive_type

Supplied parameter. Specifies how subsequent Prepare_To_Receive calls will be executed. Possible values are:

CM_PREP_TO_RECEIVE_CONFIRM

Used to send the partner program the contents of the send buffer of the logical unit (LU) and a confirmation request. Upon receipt of confirmation, the conversation changes to RECEIVE state.

CM_PREP_TO_RECEIVE_FLUSH

Used to send the partner program the contents of the local LUs send buffer and changes the conversation to RECEIVE state.

CM_PREP_TO_RECEIVE_SYNC_LEVEL

Used by the conversations synchronization level to determine prepare-to-receive processing. A default synchronization level is established by Initialize_Conversation and can be overridden by Set_Sync_Level.

If the synchronization level of the conversation is CM_NONE, the default, the contents of the local LUs send buffer are sent to the partner program and the conversation changes to RECEIVE state. If the synchronization level of the conversation is CM_CONFIRM, the contents of the local LUs send buffer and a request for confirmation are sent to the partner program. The conversation changes to RECEIVE state when the partner program issues Confirmed, responding to the confirmation request.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *prepare_to_receive_type* or *conversation_ID* is invalid.
- The *prepare_to_receive_type* parameter is set to CM_PREP_TO_RECEIVE_CONFIRM, but the conversations synchronization level is set to CM_NONE.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Set_Processing_Mode

The **Set_Processing_Mode** call (function name **cmspm**) specifies for the conversation whether subsequent calls will be returned when the operation they have requested is complete (blocking) or immediately after the operation is initiated (nonblocking).

Note

A program is notified of the completion of nonblocking calls when it issues Wait_For_Conversation or through a Microsoft® Windows® message sent to a WndProc identified by the *hWnd* in the Specify_Windows_Handle call.

Syntax

```
CM_ENTRY Set_Processing_Mode(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *receive_type,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

receive_type

Supplied parameter. Specifies whether subsequent calls on the conversation will be blocking or nonblocking. Possible values are:

CM_BLOCKING

Subsequent calls will return only when the operation is complete.

CM_NON_BLOCKING

Subsequent calls will return immediately after the operation has been initiated.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the previous incomplete operation on the conversation has not yet completed.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* or *processing_mode* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Set_Receive_Type

The **Set_Receive_Type** call (function name **cmsrt**) specifies how the program will receive data on subsequent Receive calls. It overrides the default receive type established by the Initialize_Conversation or Accept_Conversation call. By default, the program waits for data to arrive if it is not available when the **Receive** call is issued.

The receive type value affects all subsequent **Receive** calls. It can be changed by reissuing **Set_Receive_Type**.

Syntax

```
CM_ENTRY Set_Receive_Type(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *receive_type,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

receive_type

Supplied parameter. Specifies how data is to be received by the program on the subsequent Receive calls. Possible values are:

CM_RECEIVE_AND_WAIT

The local program receives any data currently available from the partner program. If no data is available, the local program waits for data to arrive.

CM_RECEIVE_IMMEDIATE

The local program receives any data currently available from the partner program. If no data is available, the local program does not wait.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* or *receive_type* is invalid, or the address of a variable is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Set_Return_Control

The **Set_Return_Control** call (function name **cmsrc**) is issued by the invoking program to specify when the local logical unit (LU), acting on the session request from the local programs Allocate call, should return control to the local program.

Syntax

```
CM_ENTRY Set_Return_Control(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *return_control,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

return_control

Supplied parameter. Specifies when the local LU, acting on the Allocate call, should return control to the local program. The following are allowed values:

CM_IMMEDIATE

The LU allocates a contention-winner session, if one is immediately available, and returns control to the program.

CM_WHEN_SESSION_ALLOCATED

The LU does not return control to the program until it allocates a session or encounters errors. If a session is not available, the program waits for one. (If the session limit is zero, the LU returns control immediately.)

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* or *return_control* is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Remarks

This call overrides the default return control established by Initialize_Conversation. By default, control is returned when the session is allocated. This call cannot be issued after the Allocate call has been issued.

For further information about sessions, see Writing CPI-C Applications.

If the LU is unable to allocate a session, the notification is returned on the **Allocate** call.

Set_Send_Type

The **Set_Send_Type** call (function name **cmsst**) specifies how data will be sent by the next Send_Data call. It overrides the default send type established by Initialize_Conversation or Accept_Conversation. The default send type is CM_BUFFER_DATA, indicating that data only (and no control information) is to be sent.

The *send_type* value affects all subsequent **Send_Data** calls. It can be changed by reissuing **Set_Send_Type**.

Syntax

```
CM_ENTRY Set_Send_Type(
  unsigned char FAR *conversation_ID,
    CM_INT32 FAR *send_type,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

send_type

Supplied parameter. Specifies how data is sent by the next Send_Data call. Possible values are:

CM_BUFFER_DATA

The data pointed to by **Send_Data** is stored in a buffer until the buffer fills up or is flushed.

CM_SEND_AND_FLUSH

The data pointed to by **Send_Data** is to be sent immediately.

CM_SEND_AND_CONFIRM

The data is to be sent immediately with a request for confirmation.

CM_SEND_AND_PREP_TO_RECEIVE

The data is to be sent immediately along with notification to the partner program that the conversation state for the sending program is changing to RECEIVE.

CM_SEND_AND_DEALLOCATE

The data is to be sent immediately along with deallocation notification.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* or *send_type* is invalid.
- The *send_type* parameter is set to CM_SEND_AND_CONFIRM, but the conversations synchronization level is set to CM_NONE.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation can be in any state except RESET.

There is no state change.

Remarks

The *send_type* values that cause additional information to be sent with the data pointed to by Send_Data let you economize on the number of calls issued. The following table summarizes **Send_Data** equivalences.

Send_Data with *send_type* set to this value	**Equates to Send_Data with *send_type* set to CM_BUFFER_DATA followed by**
CM_SEND_AND_FLUSH	Flush
CM_SEND_AND_CONFIRM	Confirm
CM_SEND_AND_PREP_TO_RECEIVE	Prepare_To_Receive
CM_SEND_AND_DEALLOCATE	Deallocate

Set_Sync_Level

The **Set_Sync_Level** call (function name **cmssl**) is issued by the invoking program to specify the synchronization level of the conversation. The synchronization level determines whether the programs synchronize their processing through the Confirm and Confirmed calls.

This call overrides the synchronization level established by the Initialize_Conversation call. The default synchronization level is CM_NONE, indicating no synchronization. This call cannot be issued after the Allocate call has been issued.

Syntax

```
CM_ENTRY Set_Sync_Level(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *sync_level,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by **Initialize_Conversation**.

sync_level

Supplied parameter. Specifies the synchronization level of the conversation. Possible values are:

CM_NONE

The programs will not perform confirmation processing.

CM_CONFIRM

The programs can perform confirmation processing.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* or *sync_level* is invalid.
- The *sync_level* parameter specifies CM_NONE but one of the following has occurred: the *send_type* parameter is set to CM_SEND_AND_CONFIRM, the *prepare_to_receive_type* parameter is set to CM_PREP_TO_RECEIVE_CONFIRM, or the *deallocate_type* is set to CM_DEALLOCATE_CONFIRM.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Set_TP_Name

The **Set_TP_Name** call (function name **cmstpn**) is issued by the invoking program to specify the partner (invokable) program name. This call overrides the partner program name derived from the side information when the Initialize_Conversation call was issued. This call cannot be issued after the Allocate call has been issued. Issuing this call has no effect on the side information itself.

Syntax

```
CM_ENTRY Set_TP_Name(
  unsigned char FAR *conversation_ID,
    unsigned char FAR *TP_name,
    CM_INT32 FAR *TP_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation.

TP_name

Supplied parameter. Specifies the starting address of the partner program name. The program name can contain up to 64 ASCII characters. The allowed characters are:

- Uppercase and lowercase letters.
- Numerals from 0 through 9.
- Special characters, except the space.

You cannot use **Set_TP_Name** to specify the name of an SNA service transaction program (TP). You can, however, use Set_CPIC_Side_Information to do this.

Double-byte character sets, such as Kanji, are not supported.

TP_name_length

Supplied parameter. Specifies the length of the partner program name. The range is from 1 through 64.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is not in INITIALIZE state.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The value specified by *conversation_ID* is invalid.
- The value specified by *TP_name_length* is out of range (greater than 64 or less than 1).
- The address of a variable is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in INITIALIZE state.

There is no state change.

Specify_Local_TP_Name

The **Specify_Local_TP_Name** call (function name **cmsltp**) is issued by the program to indicate that it is able to accept incoming conversations that are directed to the name given.

Syntax

```
CM_ENTRY Specify_Local_TP_Name(
  unsigned char FAR *TP_name,
  CM_INT32 FAR *TP_name_length,
  CM_INT32 FAR *return_code
);
```

Parameters

TP_name

Supplied parameter. Specifies the starting address of the local transaction program (TP) name. The program name can contain up to 64 ASCII characters. The allowed characters are:

- Uppercase and lowercase letters.
- Numerals from 0 through 9.
- Special characters, except the space.

You cannot use **Specify_Local_TP_Name** to specify the name of an SNA service TP.

Double-byte character sets, such as Kanji, are not supported.

TP_name_length

Supplied parameter. Specifies the length of the local program name. The range is from 1 through 64.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; one of the following occurred:

- The *TP_name* supplied is invalid.
- The value specified by *TP_name_length* is out of range (greater than 64 or less than 1).

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The call is not associated with a particular conversation, and no state restrictions apply.

There is no state change.

Remarks

A program can issue this call more than once to handle incoming conversations with more than one TP name. The program can discover the actual name on the incoming conversation by calling Extract_TP_Name.

Specify_Windows_Handle

The **Specify_Windows_Handle** call (function name **xchwnd**) sets the Microsoft® Windows® handle to which a message is sent on completion of an operation in nonblocking mode.

Syntax

```
CM_ENTRY Specify_Windows_Handle(
  HWND hwndNotify,
  CM_INT32 FAR *return_code
);
```

Parameters

hwndNotify

Supplied parameter. Specifies the Windows handle to be notified when the outstanding operation completes.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

The Windows handle is invalid.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The state change is dependent on the operation that completed and its return code.

Remarks

An application can set the processing mode by calling Set_Processing_Mode. If the Windows handle is set to NULL, or this call is never issued, the application must call Wait_For_Conversation to be notified when the outstanding operation completes.

When an asynchronous operation is complete, the applications window *hwndNotify* receives the message returned by **RegisterWindowMessage** with "WinAsyncCPIC" as the input string. The *wParam* value contains the *conversation_return_code* from the operation that is completing. Its values will depend on which operation was originally issued. The *lParam* argument contains the CM_PTR to the *conversation_ID* specified in the original function call.

Test_Request_To_Send_Received

The **Test_Request_To_Send_Received** call (function name **cmtrts**) determines whether a request-to-send notification has been received from the partner program.

Syntax

```
CM_ENTRY Test_Request_To_Send_Received(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *request_to_send_received,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Supplied parameter. Specifies the identifier for the conversation. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

request_to_send_received

Returned parameter. The request-to-send-received indicator. Possible values are:

CM_REQ_TO_SEND_RECEIVED

The partner program issued Request_To_Send, which requests the local program to change the conversation to RECEIVE state.

CM_REQ_TO_SEND_NOT_RECEIVED

The partner program did not issue **Request_To_Send**. This value is not relevant if *return_code* contains a value other than CM_OK.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_PROGRAM_PARAMETER_CHECK

Primary return code; the value specified by *conversation_ID* is invalid, or the address of a variable is invalid.

CM_PROGRAM_STATE_CHECK

Primary return code; the conversation is in a state other than SEND, RECEIVE, or SEND_PENDING.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error occurred and has been logged in the products error log.

State Changes

The conversation must be in SEND, RECEIVE, or SEND_PENDING state.

There is no state change.

Wait_For_Conversation

The **Wait_For_Conversation** call (function name **cmwait**) waits for an operation to complete that has been initiated when the *processing_mode* conversation characteristic was set to CM_NON_BLOCKING and CM_OPERATION_INCOMPLETE was returned in the *return_code* parameter.

Syntax

```
CM_ENTRY Wait_For_Conversation(
  unsigned char FAR *conversation_ID,
  CM_INT32 FAR *conversation_return_code,
  CM_INT32 FAR *return_code
);
```

Parameters

conversation_ID

Returned parameter. Specifies the identifier for the conversation on which the operation completed. The value of this parameter was returned by Initialize_Conversation or Accept_Conversation.

conversation_return_code

Returned parameter. Specifies the *return_code* from the operation that is completing. Its values will depend on which operation was originally issued.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

Primary return code; the call executed successfully.

CM_SYSTEM_EVENT

Primary return code; the wait completed not because the operation completed but because some system event occurred.

CM_PROGRAM_STATE_CHECK

Primary return code; the program has no incomplete operation outstanding.

CM_PRODUCT_SPECIFIC_ERROR

Primary return code; a product-specific error has occurred and has been logged in the products error log.

State Changes

The state change is dependent on the operation that completed and its return code.

Remarks

The program must have an incomplete operation outstanding on some conversation.

See Also

Reference

Set_Processing_Mode

Specify_Windows_Handle

CPI-C Functions Not Supported

The Microsoft® Windows® CPI-C implementation does not support the following CPI-C 1.2 functions.

CPI-C Function	Function Name
Extract_Conversation_Context	cmectx
Extract_Maximum_Buffer_Size	cmembs
Extract_Secondary_Information	cmesi
Extract_Send_Receive_Mode	cmesrm
Extract_TP_ID	xceti
Initialize_Conversation_For_TP	xcinct
Initialize_For_Incoming	cminic
Receive_Expedited	cmrcvx
Release_Local_TP_Name	cmrltp
Send_Expedited	cmsndx
Set_Queue_Callback_Function	cmsqcf
Set_Queue_Processing_Mode	cmsqpm
Set_Send_Receive_Mode	cmssrm
Start_TP	xcstp
Wait_For_Completion	cmwcmp
End_TP	xcendt

Extensions for the Windows Environment

This section describes API extensions to Microsoft® Windows® Common Programming Interface for Communications (CPI-C) that allow nonblocking or asynchronous verb completion. Asynchronous verbs return control to the program immediately, without waiting for full execution, and must notify the application later when the verb has been completed. An application is also notified in response to the completion of a Wait_For_Conversation call. In contrast, synchronous verbs block, that is, the function call does not return until the call has completed.

Under Microsoft® Windows Server™ 2003 and Windows 2000, two methods are available for handling asynchronous verb completion:

- Message posting using window handles.
- Waiting on Win32® events.

The first method uses messages posted to a window handle to notify an application of verb completion. There is one such window for each CPI-C application. Each CPI-C conversation can have one asynchronous verb outstanding at any time. When a verb completes, the posting to the window takes as parameters the CPI-C conversation identifier of the verb that has completed, and the return code of the verb.

Note
The extensions using window handles and message posting described in this section were designed for all implementations and versions of Microsoft Windows. They are now supported only for Windows Server2003 and Windows2000.

A second method using Win32 events for notification is supported on Microsoft® Host Integration Server. The extensions using Win32 events described in this section (WinCPICSetEvent and WinCPICExtractEvent)operate only on Windows Server 2003 and Windows 2000 and offer the optimum application performance in the 32-bit operating environment. If an event has been registered with the conversation, an application can call the Win32 **WaitForSingleObject** or **WaitForMultipleObjects** function to wait to be notified of the completion of the verb.

Windows CPI-C allows multithreaded Windows-based processes. Multithreading is the running of several processes in rapid sequence within a single program. A process contains one or more threads of execution.

The extension descriptions in this section provide a definition of the function, syntax, return values, and remarks for using these Windows extensions in CPI-C programs.

In This Section

- WinCPICCleanup
- WinCPICExtractEvent
- WinCPICIsBlocking
- WinCPICSetBlockingHook
- WinCPICSetEvent
- WinCPICStartup
- WinCPICUnhookBlockingHook

WinCPICCleanup

The **WinCPICCleanup** function terminates and deregisters an application from a Microsoft® Windows® Common Programming Interface for Communications (CPI-C) implementation.

Syntax

```
BOOL WINAPI WinCPICCleanup(void);
```

Return Value

The return value specifies whether the deregistration was successful. If the value is not zero, the application was successfully deregistered. The application was not deregistered if a value of zero is returned.

Remarks

Use **WinCPICCleanup** to indicate deregistration of a Windows CPI-C application from a Windows CPI-C implementation.

WinCPICExtractEvent

The **WinCPICExtractEvent** function provides a method for an application to determine the event handle being used for a Microsoft® Windows® Common Programming Interface for Communications (CPI-C) conversation.

Syntax

```
            VOID WINAPI WinCPICExtractEvent(
    unsigned char FAR*conversation_ID,HANDLE FAR*event_handle,    CM_INT32 FAR*return_code)
;
```

Parameters

conversation_ID

Specifies the identifier for the conversation for which this event is used. This parameter is returned by the initial Accept_Conversation call.

event_handle

Returned parameter. The handle of the event being used by this conversation. If no handle has been registered, this parameter returns as a NULL.

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

The function executed successfully.

CM_PROGRAM_PARAMETER_CHECK

One or more of the parameters passed to this function are invalid.

Remarks

When a verb is issued on a nonblocking conversation, it returns CM_OPERATION_INCOMPLETE if it is going to complete asynchronously. If an event has been registered with the conversation, the application can call **WaitForSingleObject** or **WaitForMultipleObjects** to be notified of the completion of the verb. **WinCPICExtractEvent** allows a CPI-C applicationto determine this event handle. When the verb has completed, the application must call Wait_For_Conversationto determine the return code for the asynchronous verb. The Cancel_Conversationfunction can be called to cancel an operation and conversation.

If no event has been registered, the asynchronous verb completes as it does at present, which is by posting a message to the window that the application has registered with the CPI-C library.

WinCPICIsBlocking

The **WinCPICIsBlocking** function determines if a task is executing while waiting for a previous blocking call to finish.

Syntax

```
BOOL WINAPI WinCPICIsBlocking(void);
```

Return Value

The return value specifies the outcome of the function. If the value is not zero, there is an outstanding blocking call awaiting completion. A value of zero indicates the absence of an outstanding blocking call.

Remarks

This call does not infer any information about a particular conversation; it is only intended to provide help to an application written to use the CM_BLOCKING characteristic of Set_Processing_Mode. **WinCPICIsBlocking** serves the same purpose as **InSendMessage** in the Microsoft® Windows® API. Legacy applications targeted at Windows version 3.*x* that support multiple conversations must specify CM_NONBLOCKING in **Set_Processing_Mode** so they can support multiple outstanding operations simultaneously. Applications are still limited to one outstanding operation per conversation in all environments.

Although a call issued on a blocking function appears to an application as though it blocks, the Windows CPI-C dynamic-link library (DLL) has to relinquish the processor to allow other applications to run. This means that it is possible for the application that issued the blocking call to be re-entered, depending on the messages it receives. In this instance, **WinCPICIsBlocking** can be used to determine whether the application task currently has been re-entered while waiting for an outstanding blocking call to finish. Note that Windows CPI-C prohibits more than one outstanding blocking call per thread.

See Also

Reference

Specify_Windows_Handle
WinCPICSetBlockingHook
WinCPICUnhookBlockingHook

WinCPICSetBlockingHook

The **WinCPICSetBlockingHook** function allows a Microsoft® Windows® Common Programming Interface for Communications (CPI-C) implementation to block CPI-C function calls by means of a new function. This legacy call was used by Microsoft® Windows® version 3.*x* applications to make blocking calls without blocking the rest of the system. By default in the Microsoft® Windows Server™ 2003 or Windows 2000 system, blocking calls suspend the calling applications thread until the request is finished.

Parameters
lpBlockFunc

Specifies the procedure instance address of the blocking function to be installed.

Return Values

The return value points to the procedure instance of the previously installed blocking function. The application or library that calls **WinCPICSetBlockingHook** should save this return value so that it can be restored if needed. (If nesting is not important, the application can simply discard the value returned by **WinCPICSetBlockingHook** and eventually use WinCPICUnhookBlockingHook to restore the default mechanism.)

Remarks

A Windows CPI-C implementation has a default mechanism by which blocking CPI-C functions are implemented. This function gives the application the ability to execute its own function at blocking time in place of the default function.

The default blocking function is equivalent to:

```
BOOL DefaultBlockingHook (void)  {
    MSG msg;
    /* get the next message if any */
    if ( PeekMessage (&msg,0,0,PM_NOREMOVE)  )  {
        if ( msg.message = = WM_QUIT  )
            return FALSE;   // let app process WM_QUIT
        PeekMessage (&msg,0,0,PM_REMOVE) ;
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    /* TRUE if no WM_QUIT received */
    return TRUE;
}
```

The **WinCPICSetBlockingHook** function is provided to support applications that require more complex message processing, for example, those employing the multiple document interface (MDI) model or applications with Menu accelerators (TranslateAccelerator).

Blocking functions must return FALSE in response to a WM_QUIT message so Windows CPI-C can return control to the application to process the message and terminate gracefully. Otherwise, the function should return TRUE.

See Also
Reference
Set_Processing_Mode
Specify_Windows_Handle

WinCPICSetEvent

The **WinCPICSetEvent** function associates an event handle with a verb completion.

Syntax

```
        VOID WINAPI WinCPICSetEvent(
unsigned char FAR* conversation_ID,HANDLE FAR* event_handle,
CM_INT32 FAR*return_code);
```

Parameters

conversation_ID

Specifies the identifier for the conversation for which this event is used. This parameter is returned by the initial Accept_Conversation call.

event_handle

The handle of the event that is to be cleared when an asynchronous verb on the conversation completes. This parameter can replace an already defined event or remove an already defined event (by having NULL as the parameter).

return_code

The code returned from this call. The valid return codes are listed later in this topic.

Return Codes

CM_OK

The function executed successfully.

CM_PROGRAM_PARAMETER_CHECK

One or more of the parameters passed to this function are invalid.

CM_OPERATION_NOT_ACCEPTED

This value indicates that a previous operation on this conversation is incomplete and the **WinCPICSetEvent** call was not accepted.

Remarks

When a verb is issued on a nonblocking conversation, it returns CM_OPERATION_INCOMPLETE if it is going to complete asynchronously. If an event has been registered with the conversation, the application can call **WaitForSingleObject** or **WaitForMultipleObjects** to be notified of the completion of the verb. When the verb has completed, the application must call Wait_For_Conversationto determine the return code for the asynchronous verb.

It is the responsibility of the application to reset the event, as it is with other APIs.

See Also

Reference

Cancel_Conversation

WinCPICStartup

The **WinCPICStartup** function allows an application to specify the version of Microsoft Windows Common Programming Interface for Communications (CPI-C) required and to retrieve details of the specific Windows CPI-C implementation. This function must be called by an application to register itself with a Windows CPI-C implementation before issuing any further Windows CPI-C calls.

Syntax

```
          INT WINAPI WinCPICStartup(
    WORDwVersionRequired,
        LPWCPICDATAlpwcpicdata);
```

Parameters

wVersionRequired

Specifies the version of Windows CPI-C support required. The high-order byte specifies the minor version (revision) number. The low-order byte specifies the major version number.

lpwcpicdata

A pointer to the CPI-C data structure. The **CPICDATA** structure is defined as follows:

```
typedef struct {
....WORD wVersion;
    char szDescription[WCPICDESCRIPTION_LEN+1];
}  CPICDATA, FAR * LPWCPICDATA;
```

WCPIDESCRIPTION is defined to be 127 and the structure members are as follows:

wVersion

The version of Windows CPI-C supported. The high-order byte specifies the minor version (revision) number. The low-order byte specifies the major version number.

szDescription

The description string describing the CPI-C version supported.

Return Value

The return value specifies whether the application was registered successfully and whether the Windows CPI-C implementation can support the specified version number. If the value is zero, it was registered successfully. Otherwise, the return value is one of the following:

WCPICSYSNOTRERADY

The underlying network system is not ready for network communication.

WCPICVERNOTSUPPORTED

The version of Windows CPI-C support requested is not provided by this particular Windows CPI-C implementation.

WCPICINVALID

The Windows CPI-C version specified by the application is not supported by this dynamic-link library (DLL).

Remarks

To support future Windows CPI-C implementations and applications that may have functionality differences from Windows CPI-C version 1.0, a negotiation takes place in **WinCPICStartup**. An application passes to **WinCPICStartup** the Windows CPI-C version that it can use. If this version is lower than the lowest version supported by the Windows CPI-C DLL, the DLL cannot support the application and the **WinCPICStartup** call fails. If the version is not lower, however, the call succeeds and returns the highest version of Windows CPI-C supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows CPI-C DLL on the system.

This negotiation allows both a Windows CPI-C DLL and a Windows CPI-C application to support a range of Windows CPI-C versions. An application can successfully use a DLL if there is any overlap in the versions. The following table illustrates how **WinCPICStartup** works in conjunction with different application and DLL versions.

Application versions	DLL versions	To WinCPICStartup	From WinCPICStartup	Result
1.0	1.0	1.0	1.0	Use 1.0
1.0, 2.0	1.0	2.0	1.0	Use 1.0
1.0	1.0, 2.0	1.0	2.0	Use 1.0
1.0	2.0, 3.0	1.0	WCPICINVALID	Fail
2.0, 3.0	1.0	3.0	1.0	App Fails
1.0, 2.0, 3.0	1.0, 2.0, 3.0	3.0	3.0	Use 3.0

Details of the actual Windows CPI-C implementation are described in the **WHLLDATA** structure defined as follows:

```
typedef struct tagWCPICDATA { WORD wVersion;
            char szDescription[WHLLDESCRIPTION_LEN+1];
            } WCPICDATA, FAR *LPWCPICDATA;
```

Having made its last Windows CPI-C call, an application should call the WinCPICCleanuproutine.

Each Windows CPI-C implementation must make a **WinCPICStartup** call before issuing any other Windows CPI-C calls.

WinCPICUnhookBlockingHook

The **WinCPICUnhookBlockingHook** function removes any previous blocking hook that has been installed and reinstalls the default blocking mechanism.

Syntax

```
BOOL WINAPI WinCPICUnhookBlockingHook(void);
```

Return Value

The return value specifies the outcome of the function. It is not zero if the default mechanism is successfully reinstalled. The value is zero if the mechanism did not reinstall.

See Also

Reference

WinCPICSetBlockingHook

CPI-C Common Return Codes

This section describes the return codes for Common Programming Interface for Communications (CPI-C) calls. The return codes are listed in integer order.

Call-specific return codes are described for the individual calls in CPI-C Calls.

0
CM_OK

The call executed successfully.

1
CM_ALLOCATION_FAILURE_NO_RETRY

The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not retry the allocation until the error has been corrected.

2
CM_ALLOCATION_FAILURE_RETRY

The conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation.

3
CM_CONVERSATION_TYPE_MISMATCH

The partner LU or program does not support the conversation type (basic or mapped) specified in the allocation request.

5
CM_PIP_NOT_SPECIFIED_CORRECTLY

The allocation request was rejected by a non-CPI-C LU 6.2 transaction program (TP). The partner program requires one or more PIP data variables, which are not supported by CPI-C.

6
CM_SECURITY_NOT_VALID

The user identifier or password specified in the allocation request was not accepted by the partner logical unit (LU).

8
CM_SYNC_LVL_NOT_SUPPORTED_PGM

The partner program does not support the synchronization level specified in the allocation request.

9
CM_TPN_NOT_RECOGNIZED

The partner LU does not recognize the program name specified in the allocation request.

10
CM_TP_NOT_AVAILABLE_NO_RETRY

The partner LU cannot start the program specified in the allocation request because of a permanent condition. The reason for the error may be logged on the remote node. Do not retry the allocation until the error has been corrected.

11
CM_TP_NOT_AVAILABLE_RETRY

The partner LU cannot start the program specified in the allocation request because of a temporary condition. The reason for the error may be logged on the remote node. Retry the allocation.

17
CM_DEALLOCATED_ABEND

The conversation has been deallocated for one of the following reasons:

- The remote program issued Deallocate with the type parameter set to CM_DEALLOCATE_ABEND. If the conversation for the remote program was in RECEIVE state when the call was issued, information sent by the local program and not

yet received by the remote program is purged.

- The partner program terminated normally but did not deallocate the conversation before terminating.

18
CM_DEALLOCATED_NORMAL

This return code does not indicate an error.

The partner program issued the Deallocate call with *deallocate_type* set to one of the following:

- CM_DEALLOCATE_FLUSH.
- CM_DEALLOCATE_SYNC_LEVEL with the synchronization level of the conversation specified as CM_NONE.

19
CM_PARAMETER_ERROR

The local program specified an invalid argument in one of its parameters.

20
CM_PRODUCT_SPECIFIC_ERROR

A product-specific error occurred and has been logged in the products error log.

21
CM_PROGRAM_ERROR_NO_TRUNC

While in SEND state or in SEND-PENDING state with the error direction set to CM_SEND_ERROR, the partner program issued Send_Error. Data was not truncated.

22
CM_PROGRAM_ERROR_PURGING

One of the following occurred:

- While in RECEIVE or CONFIRM state, the partner program issued Send_Error. Data sent but not yet received is purged.
- While in SEND-PENDING state with the error direction set to CM_RECEIVE_ERROR, the partner program issued **Send_Error**. Data was not purged.

23
CM_PROGRAM_ERROR_TRUNC (for a basic conversation)

In SEND state, before finishing sending a complete logical record, the partner program issued Send_Error. The local program may have received the first part of the logical record through a Receive call.

24
CM_PROGRAM_PARAMETER_CHECK

A parameter or the address of a variable is invalid. For details, see individual calls in CPI-C Calls.

25
CM_PROGRAM_STATE_CHECK

The call was not issued in an allowed conversation state. For details, see individual calls in CPI-C Calls.

26
CM_RESOURCE_FAILURE_NO_RETRY

One of the following occurred:

- The conversation was terminated prematurely because of a permanent condition. Do not retry until the error has been corrected.
- The partner program did not deallocate the conversation before terminating normally.

27
CM_RESOURCE_FAILURE_RETRY

The conversation was terminated prematurely because of a temporary condition, such as modem failure. Retry the conversation.

28
CM_UNSUCCESSFUL

The verb issued by the local program was not executed successfully.

30
CM_DEALLOCATED_ABEND_SVC

The conversation has been deallocated for one of the following reasons:

- The partner program issued Deallocate with the type parameter set to ABEND_SVC.
- The partner program did not deallocate the conversation before terminating.

If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

31
CM_DEALLOCATED_ABEND_TIMER

The conversation has been deallocated because the partner program issued Deallocate with the type parameter set to ABEND_TIMER. If the conversation is in RECEIVE state for the partner program when this call is issued by the local program, data sent by the local program and not yet received by the partner program is purged.

32
CM_SVC_ERROR_NO_TRUNC (for a basic conversation)

While in SEND state, the partner program or partner LU issued Send_Error with the typeparameter set to SVC. Data was not truncated.

33
CM_SVC_ERROR_PURGING

While in SEND state, the partner program or partner LU issued Send_Error with the type parameter set to SVC. Data sent to the partner program may have been purged.

34
CM_SVC_ERROR_TRUNC (for a basic conversation)

While in RECEIVE or CONFIRM state, the partner program or partner LU issued Send_Error with the type parameter set to SVC before it finished sending a complete logical record. The local program may have received the first part of the logical record.

35
CM_OPERATION_INCOMPLETE

The operation has not completed and is still in progress. The program can issue Wait_For_Conversation to await the completion of the operation, or Cancel_Conversation to cancel the operation and conversation. If Specify_Windows_Handle has been called, the application should wait for notification by a windows message and not call **Wait_For_Conversation**.

36
CM_SYSTEM_EVENT

This error code is not used by Host Integration Server 2009.

37
CM_OPERATION_NOT_ACCEPTED

A previous operation on this conversation is incomplete.

LUA Programmer's Reference

This section of the Host Integration Server 2009 Developer's Guide lists the verbs, extensions, control blocks, and return codes that describe the logical unit application (LUA) programming interface.

For general information about programming for LUA, see LUA Programmer's Guide.

For sample code that uses LUA, see LUA Samples.

In This Section

LUA RUI Verbs

This section describes the Microsoft® Windows® logical unit application (LUA) Request Unit Interface (RUI) verbs. It provides the following information for each RUI verb:

- Details of the LUA verb control block (VCB) structure.
- A description of the verb and its purpose.
- Parameters (VCB structure members) supplied to and returned by LUA. The description of each parameter includes information about the valid values for that parameter.
- Interaction with other verbs.

The verb descriptions in this section include parameter values specific to each verb. For a complete description of the VCB structure for both RUI and Session Level Interface (SLI) verbs, see LUA Verb Control Blocks.

This section contains:

- RUI_BID
- RUI_INIT
- RUI_PURGE
- RUI_READ
- RUI_TERM
- RUI_WRITE

RUI_BID

The **RUI_BID** verb notifies the Request Unit Interface (RUI) application that a message is waiting to be read using RUI_READ.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_BID**:

The second syntax union describes the **LUA_SPECIFIC** member of the verb control block (VCB) used by **RUI_BID**. Other union members are omitted for clarity:

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
union LUA_SPECIFIC {
    unsigned char lua_peek_data[12];
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for RUI verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the logical unit application (LUA) VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_BID.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_BID only requires this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Microsoft® Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Host Integration Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **RUI_BID** and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of data returned in **lua_peek_data** for **RUI_BID**.

lua_data_ptr

This parameter is not used and should be set to zero.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are set for write functions and returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for the write function and returned by the read and bid functions. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

SSCP expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

lua_message_type

Returned parameter. Specifies the type of SNA message indicated to **RUI_BID**. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

The Session Level Interface (SLI) receives and responds to the BIND, CRV, and STSN requests through the LUA interface extension routines.

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that **RUI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Reserved and should be set to zero.

lua_peek_data

The union member of **LUA_SPECIFIC** used by the **RUI_BID** and SLI_BIDverbs. Returned parameter. Contains up to 12 bytes of the data waiting to be read.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_CANCELED

Primary return code; the verb did not complete successfully because it was canceled by another verb.

LUA_TERMINATED

Secondary return code; RUI_TERM was issued while this verb was pending.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BID_ALREADY_ENABLED

Secondary return code; **RUI_BID** was rejected because a previous **RUI_BID** was already outstanding. Only one **RUI_BID** can be outstanding at any one time.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record, or a parameter not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_RUI_SESSION

Secondary return code; RUI_INIT has not yet completed successfully for the LU name specified on this verb.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_INVALID_PROCESS

Secondary return code; the process that issued this verb was not the same process that issued **RUI_INIT** for this session. Only the process that started a session can issue verbs on that session.

LUA_NEGATIVE_RSP

Primary return code; LUA detected an error in the data received from the host. Instead of passing the received message to the application on RUI_READ, LUA discards the message (and the rest of the chain if it is in a chain), and sends a negative response to the host.

LUA informs the application on a subsequent **RUI_READ** or **RUI_BID** that a negative response was sent.

The secondary return code contains the sense code sent to the host on the negative response. For information about interpreting the sense code values that can be returned, see SNA Considerations Using LUA.

A zero secondary return code indicates that, following a previous RUI_WRITE of a negative response to a message in the middle of a chain, LUA has now received and discarded all messages from this chain.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session has failed because of a problem with the link service or with the host LU.

LUA_RUI_LOGIC_ERROR

Secondary return code; an internal error was detected within LUA. This error should not occur during normal operation.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

RUI_BID is used by applications that require notification that a message is waiting to be read. This allows the application to determine how it will handle the message before issuing RUI_READ.

When a message is available, **RUI_BID** returns with details of the message flow on which it was received, the message type, the TH and RH of the message, and up to 12 bytes of message data.

The main difference between **RUI_BID** and **RUI_READ** is that **RUI_BID** allows the application to check the data without removing it from the incoming message queue, so it can be left and accessed later. **RUI_READ** removes the message from the queue, so when the application reads the data it must also process it.

Note the following when using **RUI_BID**:

- RUI_INIT must complete successfully before this verb is issued.
- Only one **RUI_BID** can be outstanding at any one time.
- After **RUI_BID** has completed successfully, it can be reissued by setting **lua_flag1.bid_enable** on a subsequent RUI_READ. If the verb is reissued in this way, the application must not free or modify the storage associated with the **RUI_BID** record.
- If a message arrives from the host when **RUI_READ** and **RUI_BID** are both outstanding, **RUI_READ** completes and **RUI_BID** is left in progress.

Each message that arrives is bid only once. After **RUI_BID** indicates that data is waiting on a particular session flow, the application issues **RUI_READ** to receive the data. Any subsequent **RUI_BID** does not report data arriving on that session flow until the message that was bid has been accepted by issuing **RUI_READ**.

In general, the **lua_data_length** parameter returned on this verb indicates only the length of data in **lua_peek_data**, not the total length of data on the waiting message (except when a value of less than 12 is returned). The application should ensure that the data length on **RUI_READ** that accepts the data is sufficient to contain the message.

See Also

Reference

RUI_INIT
RUI_READ
RUI_TERM
RUI_WRITE
SLI_OPEN

RUI_INIT

The **RUI_INIT** verb transfers control of the specified logical unit (LU) to the Microsoft® Windows® logical unit application (LUA) application. **RUI_INIT** establishes a session between the system services control point (SSCP) and the specified LU.

Note
For 3270 emulator users, a Microsoft Host Integration Server extension has been added that enables you to use 3270 LUs rat her than the LUA LUs. For more information, see Remarks in this topic.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_INIT**.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_INIT.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_INIT requires this parameter.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Host Integration Server and should be zero.

lua_sid

Returned parameter. Specifies the session identifier.

lua_max_length

Not used by **RUI_INIT** and should be set to zero.

lua_data_length

Not used by **RUI_INIT** and should be set to zero.

lua_data_ptr

Not used by **RUI_INIT** and should be set to zero.

lua_post_handle

Supplied parameter. Used under Microsoft® Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **RUI_INIT** and should be set to zero.

lua_rh

Not used by **RUI_INIT** and should be set to zero.

lua_flag1

Not used by **RUI_INIT** and should be set to zero.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. This is a returned parameter for **RUI_INIT**. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

The Session Level Interface (SLI) receives and responds to the BIND, CRV, and STSN requests through the LUA interface extension routines.

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

RUI_INIT always completes asynchronously unless it returns an error such as LUA_PARAMETER_CHECK).

lua_resv56

Supplied parameter. A reserved field used by **RUI_INIT** and SLI_OPEN. All other reserved fields in the array must be left blank. For more information, see the discussion of these Host Integration Server extensions in the Remarks section.

lua_resv56[1]

Supplied parameter. Indicates whether an RUI application can access LUs configured as 3270 LUs, in addition to LUA LUs. If this parameter is nonzero, 3270 LUs can be accessed.

lua_resv56[2]

Supplied parameter. Indicates whether the RUI library will release the LU when the LU-SSCP session or connection goes away. If this parameter is nonzero, the LU will not be released.

lua_resv56[3]

Supplied parameter. Indicates whether incomplete reads are supported. If this parameter is set to a nonzero value, incomplete or truncated reads are supported. For more details, see the remarks for RUI_READ.

lua_resv56[4]

Supplied parameter. Indicates whether the RUI library will allow the application to keep hold of the LU if it is recycled at the host. If this parameter is nonzero, the application can keep hold of the LU.

lua_encr_decr_option

Field for cryptography options. On **RUI_INIT**, only the following are supported:

lua_encr_decr_option = 0

lua_encr_decr_option = 128

Values from 1 through 127 are not supported.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_CANCELED

Primary return code; the verb did not complete successfully because it was canceled by another verb.

LUA_TERMINATED

Secondary return code; RUI_TERM was issued before **RUI_INIT** completed.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; the **lua_luname** parameter did not match any LUA LU name or LU pool name in the configuration file.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record, or a parameter not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_DUPLICATE_RUI_INIT

Secondary return code; the **lua_luname** parameter specified an LU name or LU pool name already in use by this application (or for which this application already has **RUI_INIT** in progress).

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_COMMAND_COUNT_ERROR

Secondary return code, which indicates one of the following errors occurred:

The verb could not be issued because the application had already reached its maximum number of active sessions. On Windows Server 2003 or Windows 2000, an application can have as many as 15,000 sessions active at any time.

The verb specified the name of an LU pool or the name of an LU in a pool, but all the LUs in the pool are in use.

LUA_ENCR_DECR_LOAD_ERROR

Secondary return code; the verb specified a value for **lua_encr_decr_option** other than 0 or 128.

LUA_INVALID_PROCESS

Secondary return code; the LU specified by **lua_luname** is in use by another process.

LUA_LINK_NOT_STARTED

Secondary return code; the connection to the host has not been started; none of the link services it could use are active.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session failed because of a problem with the link service or with the host LU.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

This verb must be the first LUA verb issued for the session. Until this verb has completed successfully, the only other LUA verb that can be issued for this session is RUI_TERM (which terminates a pending **RUI_INIT**).

All other verbs issued on this session must identify the session using one of the following parameters from this verb:

- The session identifier, returned to the application in **lua_sid**.
- The LU name or LU pool name, supplied by the application in the **lua_luname** parameter.

RUI_INIT completes after an ACTLU message is received from the host. If necessary, the verb waits indefinitely. If an ACTLU has already been received prior to **RUI_INIT**, LUA sends a NOTIFY to the host to inform it that the LU is ready for use.

Neither ACTLU nor NOTIFY is visible to the LUA application.

After **RUI_INIT** has completed successfully, this session uses the LU for which the session was started. No other LUA session (from this or any other application) can use the LU until **RUI_TERM** is issued, or until an LUA_SESSION_FAILURE primary return code is received.

Using 3270 LUs

To provide 3270 emulator users the ability to use the Emulator Interface Specification (EIS) configuration call with the RUI API, a Host Integration Server extension has been added to the RUI. This extension allows you to use 3270 LUs rather than LUA LUs. If an application sets **lua_resv56[1]** to a nonzero value on the **RUI_INIT** call, 3270 LUs can be used.

Do Not Release the LU

If an application sets **lua_resv56[2]** to a nonzero value on the **RUI_INIT** call, the RUI library will not release the LU when the LU-SSCP session or connection goes away. When this Host Integration Server extension is enabled, the application does not have to issue a new **RUI_INIT** after a session failure or connection failure. When the LU-SSCP session comes back up (the application can use **WinRUIGetLastInitStatus** to detect this), the application can start using it again.

Support Chunking on this Session

If an application sets **lua_resv56[3]** to a nonzero value on the **RUI_INIT** session establishment, this enables a Host Integration Server extension that can change the behavior of **RUI_READ**. The default behavior for an **RUI_READ** call is to truncate data (discarding any data remaining) if the application's data buffer is not large enough to receive all of the data in the RU, returning an error code. When **lua_resv56[3]** is set to a nonzero value on the **RUI_INIT** call, an **RUI_READ** issued where the application's data buffer is not large enough will not result in the RU data being discarded. The **RUI_READ** verb will return success (LUA_OK) for the primary return code and LUA_DATA_INCOMPLETE for the secondary return code. Subsequent **RUI_READ** requests can then be issued to retrieve the data that exceeded the application's data buffer.

Ignore DACTLUs

If an application sets **lua_resv56[4]** to a nonzero value on the **RUI_INIT** session establishment, this enables a Host Integration Server extension, and the RUI library will allow the application to keep hold of the LU if it is recycled at the host (that is,

deactivated and reactivated).

Note
All other reserved fields must be left blank.

For more information, see the description of the sepdcrec function in the section of the Software Development Kit (SDK) Help on the 3270 Emulator Interface Specification.

Encryption

Session-level cryptography is implemented through Cryptography Verification (CRV) requests. RUI applications must perform all necessary processing of these requests. For all interfaces other than RUI, CRV requests are rejected with a negative response by Host Integration Server.

For **RUI_INIT**, the following options are supported:

- **lua_encr_decr_option** = 0
- **lua_encr_decr_option** = 128

Values from 1 through 127 (ACSRENCR and ACSROECR routines) are not supported.

The sending application is responsible for padding data to a multiple of eight bytes and for setting the padded data indicator bit in the RH as well as for encryption. The receiving application is responsible for removing the padding after decryption.

See Also

Reference

RUI_INIT
RUI_TERM
SLI_OPEN

RUI_PURGE

The **RUI_PURGE** verb cancels a previous RUI_READ.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_PURGE**.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *      lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the logical unit application (LUA) VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_PURGE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_PURGE only requires this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Microsoft® Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Host Integration Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **RUI_PURGE** and should be set to zero.

lua_data_length

Not used by **RUI_PURGE** and should be set to zero.

lua_data_ptr

Points to the location of the RUI_READ verbs VCB that is to be canceled.

lua_post_handle

Supplied parameter. Used under Microsoft® Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **RUI_PURGE** and should be set to zero.

lua_rh

Not used by **RUI_PURGE** and should be set to zero.

lua_flag1

Not used by **RUI_PURGE** and should be set to zero.

lua_message_type

Not used by **RUI_PURGE** and should be set to zero.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Reserved and should be set to zero.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_CANCELED

Primary return code; the verb did not complete successfully because it was canceled by another verb.

LUA_TERMINATED

Secondary return code; RUI_TERM was issued while **RUI_PURGE** was pending.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter was set to null.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record, or a parameter not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_RUI_SESSION

Secondary return code; RUI_INIT has not yet completed successfully for the LU name specified on this verb.

LUA_UNSUCCESSFUL

Primary return code; the verb supplied was valid, but the verb did not complete successfully.

LUA_INVALID_PROCESS

Secondary return code; the OS/2 process that issued this verb was not the same process that issued **RUI_INIT** for this session. Only the process that started a session can issue verbs on that session.

LUA_NO_READ_TO_PURGE

Secondary return code; either **lua_data_ptr** did not contain a pointer to an RUI_READ VCB, or **RUI_READ** completed before **RUI_PURGE** was issued.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node was broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session failed because of a problem with the link service or with the host LU.

LUA_RUI_LOGIC_ERROR

Secondary return code; an internal error was detected within LUA. This error should not occur during normal operation.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

RUI_READ can wait indefinitely if it is sent without using the **lua_flag1.nowait** (immediate return) option and no data is available on the specified flow. **RUI_PURGE** forces the waiting verb to return (with the primary return code LUA_CANCELED).

This verb is used only when **RUI_READ** has been issued and is pending completion. (The primary return code is LUA_IN_PROGRESS.)

See Also

Reference

RUI_INIT
RUI_READ
RUI_TERM
RUI_WRITE
SLI_OPEN
SLI_PURGE
SLI_RECEIVE
SLI_SEND

RUI_READ

The **RUI_READ** verb receives responses, SNA commands, and data into a Microsoft® Windows® logical unit application (LUA) applications buffer.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_READ**.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_READ.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_READ only requires this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Specifies the length of received buffer for **RUI_READ** and SLI_RECEIVE. Not used by other RUI and SLI verbs and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of data returned in **lua_peek_data** for the RUI_BID verb.

lua_data_ptr

Pointer to the application-supplied buffer that is to receive the data from an **RUI_READ** verb. Both SNA commands and data are placed in this buffer, and they can be in an EBCDIC format.

When **RUI_READ** is issued, this parameter points to the location to receive the data from the host.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message sent or received. Various subparameters are set for write functions and returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for the write function and returned by the read and bid functions. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

SSCP expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set **lua_flag1.nowait** to 1 to indicate that you want **RUI_READ** to return immediately whether or not data is available to be read, or set it to zero if you want the verb to wait for data before returning.

Set **lua_flag1.bid_enable** to 1 to re-enable the most recent RUI_BID (equivalent to issuing **RUI_BID** again with exactly the same parameters as before), or set it to zero if you do not want to re-enable **RUI_BID**.

Re-enabling the previous **RUI_BID** reuses the VCB originally allocated for it, so this VCB must not have been freed or modified.

Set one or more of the following flags to 1 to indicate from which message flow to read data:

lua_flag1.sscp_exp

lua_flag1.lu_exp

lua_flag1.sscp_norm

lua_flag1.lu_norm

If more than one flag is set, the highest-priority data available is returned. The order of priorities (highest first) is: SSCP expedited, LU expedited, SSCP normal, LU normal. The equivalent flag in the **lua_flag2** group is set to indicate from which flow the data was read.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. Returned parameter. Specifies the type of SNA message indicated to **RUI_READ**. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that RUI_BID was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Reserved and should be set to zero.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_DATA_INCOMPLETE

Secondary return code; **RUI_READ** was not able to return all of the data received because the application's data buffer (indicated by **lua_max_length**) was not large enough. Subsequent **RUI_READ** requests can be issued to retrieve the remaining RUI data.

This is not the default behavior for **RUI_READ** and is only enabled when **lua_resv56[3]** is set to a nonzero value in the verb control block when calling RUI_INIT during session establishment. For more details, see Remarks.

LUA_CANCELED

Primary return code; the verb did not complete successfully because it was canceled by another verb or by an internal error.

LUA_PURGED

Secondary return code; **RUI_READ** has been canceled by RUI_PURGE.

LUA_TERMINATED

Secondary return code; RUI_TERM was issued while **RUI_READ** was pending.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter contained an invalid value.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BID_ALREADY_ENABLED

Secondary return code; **lua_flag1.bid_enable** was set to re-enable RUI_BID but the previous **RUI_BID** was still in progress.

LUA_DUPLICATE_READ_FLOW

Secondary return code; the flow flags in the **lua_flag1** group specified one or more session flows for which **RUI_READ** was already outstanding. Only one **RUI_READ** at a time can be waiting on each session flow.

LUA_INVALID_FLOW

Secondary return code; none of the **lua_flag1** flow flags was set. At least one of these flags must be set to 1, to indicate from which flow or flows to read.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_NO_PREVIOUS_BID_ENABLED

Secondary return code; **lua_flag1.bid_enable** was set to re-enable RUI_BID, but there was no previous **RUI_BID** that could be enabled. (For more information, see Remarks.)

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record or a parameter not used by this verb was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_RUI_SESSION

Secondary return code; RUI_INIT has not yet completed successfully for the LU name specified on **RUI_READ**.

LUA_NEGATIVE_RSP

Primary return code; indicates one of the following two cases, which can be distinguished by the secondary return code:

- LUA detected an error in the data received from the host. Instead of passing the received message to the application on **RUI_READ**, LUA discards the message (and the rest of the chain if it is in a chain), and sends a negative response to the host. LUA informs the application on a subsequent **RUI_READ** or RUI_BID that a negative response was sent.

- The LUA application previously sent a negative response to a message in the middle of a chain. LUA has purged subsequent messages in this chain, and is now reporting to the application that all messages from the chain have been received and purged.

LUA_SEC_RC

Secondary return code; this parameter is a nonzero secondary return code containing the sense code sent to the host on the negative response. This indicates that LUA detected an error in the host data and sent a negative response to the host. For information about interpreting the sense code values that may be returned, see SNA Considerations Using LUA.

A secondary return code of zero indicates that, following a previous RUI_WRITE of a negative response to a message in the middle of a chain, LUA has now received and discarded all messages from this chain.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_DATA_TRUNCATED

Secondary return code; the **lua_data_length** parameter was smaller than the actual length of data received on the message. Only **lua_data_length** bytes of data were returned to the verb; the remaining data was discarded. Additional parameters are also returned if this secondary return code is obtained.

LUA_NO_DATA

Secondary return code; **lua_flag1.nowait** was set to indicate immediate return without waiting for data, and no data was currently available on the specified session flow or flows.

LUA_INVALID_PROCESS

Secondary return code; the OS/2 process that issued this verb was not the same process that issued RUI_INIT for this session. Only the process that started a session can issue verbs on that session.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node was broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session failed because of a problem with the link service or with the host LU.

LUA_RUI_LOGIC_ERROR

Secondary return code; an internal error was detected within LUA. This error should not occur during normal operation.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

RUI_INIT must have completed successfully before **RUI_READ** is issued.

While an existing **RUI_READ** is pending, you can issue another **RUI_READ** only if it specifies a different session flow or flows from pending **RUI_READ** verbs. You cannot have more than one **RUI_READ** outstanding for the same session flow.

You can specify a particular message flow (LU normal, LU expedited, SSCP normal, or SSCP expedited) from which to read

data, or you can specify more than one message flow. You can have multiple **RUI_READ** verbs outstanding, provided that no two of them specify the same flow.

Data is received by the application on one of four session flows. The four session flows, from highest to lowest priority are:

- SSCP expedited
- LU expedited
- SSCP normal
- LU normal

The data flow type that **RUI_READ** is to process is specified in the **lua_flag1** parameter. The application can also specify whether it wants to look at more than one type of data flow. When multiple flow bits are set, the highest priority is received first. When **RUI_READ** completes processing, **lua_flag2** indicates the specific type of flow for which data has been received by the Windows LUA application.

If RUI_BID successfully completes before an **RUI_READ** is issued, the Windows LUA interface can be instructed to reuse the last **RUI_BID** verbs VCB. To do this, issue the **RUI_READ** with **lua_flag1.bid_enable** set.

The **lua_flag1.bid_enable** parameter can be used only if the following are true:

- **RUI_BID** has already been issued successfully and has completed.
- The storage allocated for **RUI_BID** has not been freed or modified.
- No other **RUI_BID** is pending.

When using **lua_flag1.bid_enable**, the **RUI_BID** storage must not be freed because the last **RUI_BID** verbs VCB is used. Also, when using **lua_flag1.bid_enable**, the successful completion of **RUI_BID** will be posted.

If **RUI_READ** is issued with **lua_flag1.nowait** when no data is available to receive, LUA_NO_DATA will be the secondary return code set by the Windows LUA interface.

If the data received is longer than **lua_max_length**, it is truncated. Only **lua_max_length** bytes of data are returned. The primary return code LUA_UNSUCCESSFUL and the secondary return code LUA_DATA_TRUNCATED are also returned. The RUI library returns as much data as possible to the application's data buffer, but the remaining data in the RUI is discarded and cannot be extracted on subsequent **RUI_READ** requests. This forces the RUI application to allocate an **RUI_READ** data buffer large enough to handle the full RU size.

This default behavior can be changed by setting the value of **lua_resv56[3]** to a nonzero value in the verb control block when calling RUI_INIT during session establishment. In this case, if the data received is longer than **lua_max_length**, an **RUI_READ** request will return a primary return code of LUA_OK and a secondary return code of LUA_DATA_INCOMPLETE. An RUI application can then issue new **RUI_READ** calls and receive the remainder of the data.

This enhancement has not been adopted as part of the Microsoft Windows Open Services Architecture (WOSA) LUA API standard and differs from the implementation of RUI by IBM.

After a message has been read using **RUI_READ,** it is removed from the incoming message queue and cannot be accessed again. (RUI_BID can be used as a nondestructive read. The application can use it to check the type of data available, but the data remains on the incoming queue and does not need to be used immediately.)

Pacing can be used on the primary-to-secondary half-session (specified in the host configuration), to protect the LUA application from being flooded with messages. If the LUA application is slow to read messages, Host Integration Server delays the sending of pacing responses to the host to slow it down.

See Also

Reference

RUI_BID
RUI_INIT
RUI_TERM
RUI_WRITE
SLI_OPEN

SLI_PURGE
SLI_RECEIVE
SLI_SEND

RUI_TERM

The **RUI_TERM** verb ends both the logical unit (LU) session and the system services control point (SSCP) session for a given LUA LU.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_TERM**.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the logical unit application (LUA) VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_TERM.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_TERM only requires this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Microsoft® Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Host Integration Server and should be set to zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **RUI_TERM** and should be set to zero.

lua_data_length

Not used by **RUI_TERM** and should be set to zero.

lua_data_ptr

Not used by **RUI_TERM** and should be set to zero.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **RUI_TERM** and should be set to zero.

lua_rh

Not used by **RUI_TERM** and should be set to zero.

lua_flag1

Not used by **RUI_TERM** and should be set to zero.

lua_message_type

Not used by **RUI_TERM** and should be set to zero.

lua_flag2

Not used by **RUI_TERM** and should be set to zero.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Reserved and should be set to zero.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record or a parameter not used by this verb was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_RUI_SESSION

Secondary return code; RUI_INIT has not yet completed successfully for the LU name specified on **RUI_TERM**.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_COMMAND_COUNT_ERROR

Secondary return code; **RUI_TERM** was already pending when the verb was issued.

LUA_INVALID_PROCESS

Secondary return code; the OS/2 process that issued this verb was not the same process that issued **RUI_INIT** for this session. Only the process that started a session can issue verbs on that session.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node was broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session failed because of a problem with the link service or with the host LU.

LUA_RUI_LOGIC_ERROR

Secondary return code; an internal error was detected within LUA. This error should not occur during normal operation.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

This verb can be issued at any time after RUI_INIT has been issued (whether or not it has completed). If any other LUA verb is pending when **RUI_TERM** is issued, no further processing on the pending verb takes place, and it returns with a primary return

code of LUA_CANCELED.

After this verb has completed, no other LUA verb can be issued for this session.

See Also

Reference

RUI_INIT

SLI_OPEN

RUI_WRITE

The **RUI_WRITE** verb sends an SNA request or response unit from the logical unit application (LUA) application to the host over either the LU session or the system services control point (SSCP) session, and sends responses, SNA commands, and data from a Microsoft® Windows® LUA application to the host LU.

The following structure describes the **LUA_COMMON** member of the verb control block (VCB) used by **RUI_WRITE**.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_RUI_WRITE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

RUI_WRITE only requires this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by RUI in Microsoft® Host Integration Server and should be set to zero.

lua_cobol_offset

Not used by LUA in Host Integration Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **RUI_WRITE** and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of data returned in **lua_peek_data** for the RUI_BID verb.

lua_data_ptr

Points to the buffer containing the data to be sent to the host by **RUI_WRITE**.

Both SNA commands and data are placed in this buffer, and they can be in an EBCDIC format.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message sent or received. Various subparameters are set for write functions and returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. For the RH for **RUI_WRITE**, all fields except the queued-response indicator (**lua_rh.qri**) and pacing indicator (**lua_rh.pi**) are used. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

SSCP expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set one of the following flags to 1 to indicate on which message flow the data is to be sent:

lua_flag1.sscp_exp

lua_flag1.sscp_norm

lua_flag1.lu_exp

lua_flag1.lu_norm

lua_message_type

Not used by **RUI_WRITE** and should be set to zero.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that RUI_BID was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Reserved and should be set to zero.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_CANCELED

Primary return code; the verb did not complete successfully because it was canceled by another verb.

LUA_TERMINATED

Secondary return code; the verb was canceled because RUI_TERM was issued for this session.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter contained an invalid value.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_DUPLICATE_WRITE_FLOW

Secondary return code; **RUI_WRITE** was already outstanding for the session flow specified on this verb (the session flow is specified by setting one of the **lua_flag1** flow flags to 1). Only one **RUI_WRITE** at a time can be outstanding on each session flow.

LUA_INVALID_FLOW

Secondary return code; the **lua_flag1.sscp_exp** flow flag was set, indicating that the message should be sent on the SSCP expedited flow. LUA does not allow applications to send data on this flow.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_MULTIPLE_WRITE_FLOWS

Secondary return code; more than one of the **lua_flag1** flow flags was set to 1. One and only one of these flags must be set to 1, to indicate which session flow the data is to be sent on.

LUA_REQUIRED_FIELD_MISSING

Secondary return code; indicates one of the following cases:

- None of the **lua_flag1** flow flags was set. One and only one of these flags must be set to 1.
- **RUI_WRITE** was used to send a response, and the response required more data than was supplied.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved field in the verb record or a parameter not used by this verb was set to a nonzero value.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_MODE_INCONSISTENCY

Secondary return code; the SNA message sent on **RUI_WRITE** was not valid at this time. This is caused by trying to send data on the LU session before the session is bound. Check the sequence of SNA messages sent.

LUA_NO_RUI_SESSION

Secondary return code; RUI_INIT has not yet completed successfully for the LU name specified on this verb.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; indicates one of the following cases:

- The **lua_rh.fi** bit (format indicator) was set to 1, but the first byte of the supplied RU was not a recognized request code.
- The **lua_rh.ruc** parameter (RU category) specified the network control (NC) category; LUA does not allow applications to send requests in this category.

LUA_INVALID_PROCESS

Secondary return code; the OS/2 process that issued this verb was not the same process that issued **RUI_INIT** for this session. Only the process that started a session can issue verbs on that session.

LUA_INVALID_SESSION_PARAMETERS

Secondary return code; the application used **RUI_WRITE** to send a positive response to a BIND message received from the host. However, Host Integration Server cannot accept the BIND parameters as specified, and has sent a negative response to the host. For more information about the BIND profiles accepted by Host Integration Server, see SNA Considerations Using LUA.

LUA_RSP_CORRELATION_ERROR

Secondary return code; when using **RUI_WRITE** to send a response, **lua_th.snf** (which indicates the sequence number of the received message being responded to) did not contain a valid value.

LUA_RU_LENGTH_ERROR

Secondary return code; the **lua_data_length** parameter contained an invalid value. When sending data on the LU normal flow, the maximum length is as specified in the BIND received from the host; for all other flows the maximum length is 256 bytes.

Note
Any other secondary return code is an SNA sense code indicating that the supplied SNA data was invalid or could not be sent. For information about interpreting the SNA sense codes that can be returned, see SNA Considerations Using LUA.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node was broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_SESSION_FAILURE

Primary return code; a required Host Integration Server component has terminated.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; indicates that the LUA session has failed because of a problem with the link service or with the host LU.

LUA_RUI_LOGIC_ERROR

Secondary return code; an internal error was detected within LUA. This error should not occur during normal operation.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or has terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

Remarks

RUI_INIT must be issued successfully before this verb is issued.

When sending an SNA request, all applicable values in the **lua_rh** must be set. Chaining and bracketing are the responsibility of the application.

When sending a response, the type of response determines the **RUI_WRITE** information required. For all responses, you must:

- Set the selected **lua_rh.rri** flag to 1.
- Provide the sequence number in **lua_th.snf** for the request to which you are responding.

For multi-chain message responses, the sequence number of the last received chain element must be used. For a response to a multichain message ending with a CANCEL command, the CANCEL command sequence number is used.

For positive responses that only require the request code, set **lua_rh.ri** to zero (indicating that the response is positive) and **lua_data_length** to zero (indicating that no data is provided). The request code is filled in by the RUI, using the sequence number provided.

For negative responses, set **lua_rh.ri** to 1, **lua_data_ptr** to the SNA sense code address, and **lua_data_length** to the SNA sense code length (four bytes). The sequence number is used by the RUI to fill in the request code.

For positive responses to the BIND and STSN commands that require data in the responses, set **lua_data_ptr** to point to the response and set **lua_data_length** to the length of the data provided in **lua_data_ptr**.

While an existing **RUI_WRITE** is pending, you can issue a second **RUI_WRITE** only if it specifies a different session flow from the pending **RUI_WRITE**. You cannot have more than one **RUI_WRITE** outstanding for the same session flow.

RUI_WRITE can be issued on the SSCP normal flow at any time after a successful RUI_INIT. **RUI_WRITE** verbs on the LU expedited or LU normal flows are permitted only after a BIND has been received, and must abide by the protocols specified on the BIND.

The successful completion of **RUI_WRITE** indicates that the message was queued successfully to the data link. It does not necessarily indicate that the message was sent successfully, or that the host accepted it.

Pacing can be used on the secondary-to-primary half-session (specified on the BIND) to prevent the LUA application from sending more data than the local or remote LU can handle. If this is the case, an **RUI_WRITE** on the LU normal flow may be delayed by LUA and may take some time to complete.

See Also

Reference

RUI_INIT
RUI_READ
RUI_TERM
SLI_OPEN
SLI_PURGE
SLI_RECEIVE
SLI_SEND

LUA SLI Verbs

This section describes the Microsoft® Windows® logical unit application (LUA) Session Level Interface (SLI) verbs. It provides the following information for each SLI verb:

- Details of the LUA verb control block (VCB) structure.
- A description of the verb and its purpose.
- Parameters (VCB structure members) supplied to and returned by LUA. The description of each parameter includes information about the valid values for that parameter.
- Interaction with other verbs.

Cryptography is not defined as part of the Windows LUA standard.

The verb descriptions in this section include parameter values specific to each verb. For a complete description of the VCB structure for both Request Unit Interface (RUI) and SLI verbs, see LUA Verb Control Blocks.

In This Section

SLI_BID

The **SLI_BID** verb notifies the Session Level Interface (SLI) application that a message is waiting to be read using SLI_RECEIVE. **SLI_BID** also provides the current status of the session to the Windows logical unit application (LUA) application.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_BID.

The second syntax union describes the **LUA_SPECIFIC** member of the VCB used by **SLI_BID**. Other union members are omitted for clarity.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
};
union LUA_SPECIFIC {
    unsigned char  lua_peek_data[12];
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for SLI verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_BID.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_BID only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_BID** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_BID** and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of data returned in **lua_peek_data**.

lua_data_ptr

Pointer to the application-supplied buffer that contains the data to be sent for SLI_SEND and RUI_WRITE or that will receive data for SLI_RECEIVE and RUI_READ. Not used by other RUI and SLI verbs and should be set to zero.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

For all other environments, this parameter is reserved and should be set to zero.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

Request/response unit (RU) category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

lua_message_type

Returned parameter. Specifies the type of SNA message indicated to SLI_BID. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_STSN

The SLI receives and responds to the BIND and STSN requests through the LUA interface extension routines.

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that **SLI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_BID** and should be set to zero.

lua_peek_data

The union member of **LUA_SPECIFIC** used by the RUI_BIDand **SLI_BID** verbs. Returned parameter. Contains up to 12 bytes of the data waiting to be read.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** name was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with the value of **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_SLI_BID_PENDING

Secondary return code; an SLI verb was still active when another **SLI_BID** was issued. Only one **SLI_BID** can be active at a

time.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued, or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_RECEIVER_IN_TRANSMIT_MODE

Secondary return code; either resources needed to handle normal flow data were not available or the state of the half-duplex contention was not received when a normal-flow request was received. The result is a race condition. This SNA sense code is also an exception request sense code.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_CHAINING_ERROR

Secondary return code; the sequence of the chain indicator settings is in error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_BRACKET

Secondary return code; the sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DIRECTION

Secondary return code; while the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_QUIESCED

Secondary return code; a data flow control (DFC) or function management data (FMD) request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_NO_BEGIN_BRACKET

Secondary return code; the receiver has already sent a positive response to a BIS command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_IMMEDIATE_REQUEST_MODE_ERROR

Secondary return code; the request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_QUEUED_RESPONSE_ERROR

Secondary return code; the request violated the queued response protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_ERP_SYNC_EVENT_ERROR

Secondary return code; a violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_CORRELATION_ERROR

Secondary return code; a response was sent that does not correspond to a previously received request or a response was received that does not correspond to a previously sent request.

LUA_RSP_PROTOCOL_ERROR

Secondary return code; a violation of the response protocol was found in the response received from the primary half-session.

LUA_BB_NOT_ALLOWED

Secondary return code; the begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EB_NOT ALLOWED

Secondary return code; the end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EXCEPTION_RSP_NOT_ALLOWED

Secondary return code; when an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_DEFINITE_RSP_NOT_ALLOWED

Secondary return code; when a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_ALLOWED

Secondary return code; the change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_RESPONSE_NOT_ALLOWED

Secondary return code; a request other than an EXR contained a "NO RESPONSE" The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CHAINING_NOT_SUPPORTED

Secondary return code; the chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was

prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_BRACKETS_NOT_SUPPORTED

Secondary return code; the bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_SUPPORTED

Secondary return code; the change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_FI

Secondary return code; the format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_ALTERNATE_CODE_NOT_SUPPORTED

Secondary return code; the code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_RU_CATEGORY

Secondary return code; the request unit category indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_REQUEST_CODE

Secondary return code; the request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_SPEC_OF_SDI_RTI

Secondary return code; the sense-data-included indicator (SDI) and the response-type-indicator (RTI) were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_DR1I_DR2I_ERI

Secondary return code; the definite response 1 indicator (DR1I), the definite response 2 indicator (DR2I), and the exception response indicator (ERI) were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_QRI

Secondary return code; the queued response indicator (QRI) was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_EDI

Secondary return code; the enciphered data indicator (EDI) was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was

prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_PDI

Secondary return code; the padded data indicator (PDI) was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid, but the verb did not complete successfully.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server when an LUA verb was issued.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_RECEIVE_CORRELATION_TABLE_FULL

Secondary return code; the session receive correlation table for the flow requested reached its capacity.

LUA_NEGATIVE_RESPONSE

Primary return code; either LUA sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; the LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_DATA_TRAFFIC_RESET

Secondary return code; a half-session of an active session but with inactive data traffic received a normal flow DFC or FMD request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_NOT_RESET

Secondary return code; while the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_SC_PROTOCOL_VIOLATION

Secondary return code; a violation of the session control (SC) protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_INVALID_SC_OR_NC_RH

Secondary return code; the RH of an SC or NC request was invalid.

LUA_PACING_NOT_SUPPORTED

Secondary return code; the request contained a pacing indicator when support of pacing for this session does not exist for

the receiving half-session or boundary function half-session. The BIND options chosen previously or the architectural rules were violated by **lua_rh** values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NAU_INOPERATIVE

Secondary return code; the network addressable unit (NAU) is not able to process responses or requests. Delivery to the receiver could not take place for one of the following reasons:

A path information unit error

A path outage

An invalid sequence of requests for activation

If a path error is received during an active session, that usually means there is no longer a valid path to the session partner.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process has been canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_STATUS

Primary return code; the secondary return code contains SLI status information for the application.

LUA_READY

Secondary return code; following a NOT_READY status, this status is issued to notify you that the SLI is ready to process commands.

LUA_NOT_READY

Secondary return code; an SNA UNBIND type 0x02 command was received, which means a new BIND is coming.

If the UNBIND type 0x02 is received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command.

If the UNBIND type 0x02 is received while SLI_OPEN is still processing, the primary return code is session-failure, not status. Or, the receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_INIT_COMPLETE

Secondary return code; the LUA interface initialized the session while **SLI_OPEN** was processing. LUA applications that issue **SLI_OPEN** with the **lua_open_type_prim_sscp** parameter receive this status on SLI_RECEIVE or **SLI_BID**.

LUA_SESSION_END_REQUESTED

Secondary return code; the LUA interface received an SNA shutdown command (SHUTD) from the host, which means the host is ready to shut down the session.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation encountered an ABEND.

The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).

The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_BID does the following:

- Notifies a Windows LUA application that a message is waiting to be read.
- Provides the current session status.
- Provides a preview of the next message that will be read by SLI_RECEIVE.

  This preview contains a maximum of 12 bytes of information (peek data) that enables the Windows LUA application to define its processing strategy for the data.

To use **SLI_BID** within a Windows LUA application, issue **SLI_BID**. When the verb completes, it can be reactivated in the following two ways:

- Reissue **SLI_BID**.
- Issue SLI_RECEIVE with lua_flag1_bid_enable set to 1. This issues an SLI_BID that uses the most recently accepted address for the VCB and establishes the active bid.

Each session can have only one **SLI_BID** at a time.

If multiple messages are available when a Windows LUA application issues SLI_BID, the data flow with the highest priority is returned. The order in which the data can be returned is as follows:

- SSCP expedited
- LU expedited
- SSCP normal
- LU normal

If SLI_RECEIVE has flags set to read more than one type of message flow, the data returned by **SLI_BID** might be for a flow different than the one for which you actually receive data through **SLI_RECEIVE**. This situation occurs when higher priority data arrives from the host after **SLI_BID** completes processing, but before **SLI_RECEIVE** is issued.

To ensure that SLI_RECEIVE reads the data, the SLI_BID returned specifies the flow that matches lua_flag2 returned by the completed SLI_BID.

Session Status Return Values

If LUA_STATUS is the primary return code, the secondary return code can be LUA_READY, LUA_NOT_READY, LUA_SESSION_END_REQUESTED, or LUA_INIT_COMPLETE. In addition, if LUA_STATUS is the primary return code, the following parameters are used:

lua_sec_rc

lua_sid

LUA_READY is returned after LUA_NOT_READY status, and indicates that the SLI is again ready to perform all commands.

LUA_NOT_READY indicates that the SLI session is suspended because the SLI has received either an SNA CLEAR command or an SNA UNBIND command with an 0x02 UNBIND type (UNBIND with BIND forthcoming). Depending on what caused the suspension, the session can be reactivated as follows:

- When the suspension is caused by an SNA CLEAR, receiving an SNA SDT reactivates the session.
- When an SNA UNBIND type BIND forthcoming causes suspension of the session and the SLI_OPEN that opened the session is completed, the session is suspended until the SLI receives a BIND and SDT command. The session can also optionally receive an STSN command. As a result, user-supplied routines issued with the initial SLI_OPEN must be re-entered because they will be recalled.

The application can send SSCP data after a CLEAR or UNBIND type BIND forthcoming arrives and before the NOT READY status is read. The application can send and receive SSCP data after reading a NOT READY.

When an SNA UNBIND type BIND forthcoming arrives before completion of the SLI_OPEN that opened the session, LUA_SESSION_FAILURE (not LUA_STATUS) is the primary return code.

LUA_SESSION_END_REQUESTED indicates that the application received an SNA SHUTD from the host. The Windows LUA application should issue SLI_CLOSE to close the session when convenient.

LUA_INIT_COMPLETE is returned only when lua_init_type for SLI_OPEN is LUA_INIT_TYPE_PRIM_SSCP. The status means that SLI_OPEN has been processed sufficiently to allow SSCP data to now be sent or received.

Exception Requests

If a host application request unit is converted into an EXR, sense data will be returned. When an **SLI_BID** completes with the returned verb parameters set as shown, an EXR conversion occurs.

Member	**Set to**
lua_prim_rc	OK (0x0000)
lua_sec_rc	OK (0x00000000)
lua_rh.rri	bit off (request unit)
lua_rh.sdi	bit on (includes sense data)

Of the seven bytes of data in **lua_peek_data**, bytes 0 through 3 define the error detected. The following table indicates possible sense data and the values of bytes 0 through 3.

Sense data	**Value of bytes 0–3**
LUA_MODE_INCONSISTENCY	0x08090000
LUA_BRACKET_RACE_ERROR	0x080B0000
LUA_BB_REJECT_NO_RTR	0x08130000
LUA_RECEIVER_IN_TRANSMIT_MODE	0x081B0000
LUA_CRYPTOGRAPHY_FUNCTION_INOP	0x08480000
LUA_SYNC_EVENT_RESPONSE	0x10010000
LUA_RU_DATA_ERROR	0x10020000

LUA_RU_LENGTH_ERROR	0x10020000
LUA_INCORRECT_SEQUENCE_NUMBER	0x20010000

The information returned to bytes 3 through 6 in **lua_peek_data** is determined by the first 3 bytes of the initial request unit that caused the error.

See Also

Reference

RUI_INIT

SLI_CLOSE

SLI_OPEN

SLI_RECEIVE

SLI_CLOSE

The **SLI_CLOSE** verb ends a session opened with SLI_OPEN. The LU-LU and LU-SSCP resources are released.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_CLOSE.

Syntax

```
struct LUA_COMMON {
    unsigned short   lua_verb;
    unsigned short   lua_verb_length;
    unsigned short   lua_prim_rc;
    unsigned long    lua_sec_rc;
    unsigned short   lua_opcode;
    unsigned long    lua_correlator;
    unsigned char    lua_luname[8];
    unsigned short   lua_extension_list_offset;
    unsigned short   lua_cobol_offset;
    unsigned long    lua_sid;
    unsigned short   lua_max_length;
    unsigned short   lua_data_length;
    char FAR *       lua_data_ptr;
    unsigned long    lua_post_handle;
    struct LUA_TH    lua_th;
    struct LUA_RH    lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char    lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char    lua_resv56[7];
    unsigned char    lua_encr_decr_option;

};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the logical unit application (LUA) VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_CLOSE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_CLOSE only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_CLOSE** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_CLOSE** and should be set to zero.

lua_data_length

Not used by **SLI_CLOSE** and should be set to zero.

lua_data_ptr

Not used by **SLI_CLOSE** and should be set to zero.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **SLI_CLOSE** and should be set to zero.

lua_rh

Not used by **SLI_CLOSE** and should be set to zero.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit. A supplied parameter used by **SLI_CLOSE** to specify whether the session is to be closed immediately (ON) or closed normally (OFF). For verbs other than **SLI_CLOSE**, this flag must be off.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

lua_message_type

Not used by **SLI_CLOSE** and should be set to zero.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_CLOSE** and should be set to zero.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an **SLI_CLOSE** or session failure when a command was issued.

LUA_CLOSE_PENDING

Secondary return code; one of the following has occurred:

- A CLOSE_ABEND was still pending when another CLOSE_ABEND was issued. You can issue a CLOSE_ABEND if a CLOSE_NORMAL is pending.
- Either a CLOSE_ABEND or a CLOSE_NORMAL was still pending when a CLOSE_NORMAL was issued.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server when an LUA verb was issued.

LUA_UNEXPECTED_SNA_SEQUENCE

Secondary return code; unexpected data or commands were received from the host while SLI_OPEN was processing.

LUA_NEGATIVE_RSP_CHASE

Secondary return code; a negative response to an SNA CHASE command from the host was received by the LUA interface while **SLI_CLOSE** was being processed. **SLI_CLOSE** continued processing to stop the session.

LUA_NEGATIVE_RSP_SHUTC

Secondary return code; a negative response to an SNA SHUTC command from the host was received by the SLI while **SLI_CLOSE** was still being processed. **SLI_CLOSE** continued processing to stop the session.

LUA_NEGATIVE_RSP_SHUTD

Secondary return code; a negative response to an SNA RSHUTD command from the host was received by the LUA interface while **SLI_CLOSE** was still being processed. **SLI_CLOSE** continued processing to stop the session.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued, or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

There are two types of **SLI_CLOSE**: normal and ABEND. For a normal close, **lua_flag1.close_abend** is set to zero. The sequence for a normal close can be initiated either as primary (host-initiated) or secondary (requested by a Windows LUA application). During a primary normal close, the Windows LUA interface:

- Reads the SHUTD command and posts the SESSION_END_REQUESTED status to the application.
- Writes the CHASE command (if necessary).
- Reads and processes the CHASE command response (if necessary).
- Writes the shutdown complete (SHUTC) command.
- Reads and processes the SHUTC command response.
- Reads and processes the CLEAR command (if necessary).
- Writes the CLEAR command response (if necessary).
- Reads and processes the UNBIND command.
- Writes the UNBIND command response.
- Stops the session.

During a secondary normal close, the Windows LUA interface:

- Writes the RSHUTD command.
- Reads and processes the RSHUTD command response.
- Reads and processes the CLEAR command (if necessary).
- Writes the CLEAR command response (if necessary).
- Reads and processes the UNBIND command.
- Writes the UNBIND command response.
- Stops the session.

For an ABEND close, **lua_flag1.close_abend** is set to 1, which directs the Windows LUA interface to close the session immediately. After **SLI_CLOSE** starts processing, the LU-LU connection is terminated and the SSCP is informed that the LU is not capable of sustaining a session.

See Also
Reference
SLI_OPEN

SLI_OPEN

The **SLI_OPEN** verb transfers control of the specified logical unit (LU) to the Microsoft® Windows® logical unit application (LUA) application. **SLI_OPEN** establishes a session between the system services control point (SSCP) and the specified LU, as well as an LU-LU session.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_OPEN.

The second syntax union describes the **LUA_SPECIFIC** member of the VCB used by **SLI_OPEN**. Other union members are omitted for clarity.

Syntax

```
struct LUA_COMMON {
    unsigned short   lua_verb;
    unsigned short   lua_verb_length;
    unsigned short   lua_prim_rc;
    unsigned long    lua_sec_rc;
    unsigned short   lua_opcode;
    unsigned long    lua_correlator;
    unsigned char    lua_luname[8];
    unsigned short   lua_extension_list_offset;
    unsigned short   lua_cobol_offset;
    unsigned long    lua_sid;
    unsigned short   lua_max_length;
    unsigned short   lua_data_length;
    char FAR *       lua_data_ptr;
    unsigned long    lua_post_handle;
    struct LUA_TH    lua_th;
    struct LUA_RH    lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char    lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char    lua_resv56[7];
    unsigned char    lua_encr_decr_option;
};
```

```
union LUA_SPECIFIC {
    struct union SLI_OPEN open;
};

The SLI_OPEN structure contains the following nested structures and members:
struct LUA_EXT_ENTRY {
    unsigned char lua_routine_type;
    unsigned char lua_module_name[9];
    unsigned char lua_procedure_name[33];
} ;

struct SLI_OPEN {
    unsigned char         lua_init_type;
    unsigned char         lua_resv65;
    unsigned short        lua_wait;
    struct LUA_EXT_ENTRY  lua_open_extension[3];
    unsigned char         lua_ending_delim;
} ;
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_OPEN.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_OPEN requires this parameter.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Supplied parameter. Specifies the offset from the start of the VCB to the extension list of user-supplied dynamic-link libraries (DLLs). The value must be the beginning of a word boundary unless there is no extension list. In this case, the value must be set to zero.

If this option is not used by SLI_OPEN, this member should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server and should be zero.

lua_sid

Returned parameter. Specifies the session identifier.

lua_max_length

Not used by **SLI_OPEN** and should be set to zero.

lua_data_length

Supplied parameter. Specifies the actual length of the data being sent.

lua_data_ptr

Pointer to the application-supplied buffer that contains the data to be sent for **SLI_OPEN**.

Both SNA commands and data are placed in this buffer, and they can be in an Extended Binary Coded Decimal Interchange Code (EBCDIC) format.

When SLI_OPEN is issued, this parameter can be one of the following:

- The LOGON message for the SSCP normal flow when the initialization type is secondary with an unformatted LOGON message.
- The request/response unit (RU) for INITSELF. When the initialization type is secondary with INITSELF, the necessary data for the application is provided.
- For all other open types, this field should be set to zero.

This information is provided by the Windows LUA application.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is

to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **SLI_OPEN** and should be set to zero.

lua_rh

Not used by **SLI_OPEN** and should be set to zero.

lua_flag1

Not used by **SLI_OPEN** and should be set to zero.

lua_message_type

Not used by **SLI_OPEN** and should be set to zero.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_resv56

Supplied parameter. Reserved field used by **SLI_OPEN** and RUI_INIT. For more information, see the Remarks section.

lua_resv56[1]

Supplied parameter. This parameter must be set to zero.

lua_resv56[2]

Supplied parameter. Indicates whether an SLI application can access LUs configured as 3270 LUs, in addition to LUA LUs. If this parameter is set to 1, 3270 LUs can be accessed.

lua_resv56[3]

Supplied parameter. Indicates whether incomplete reads are supported. If this parameter is set to 1, incomplete or truncated reads are supported. For more details, see the remarks for RUI_READ.

lua_encr_decr_option

Not used by **SLI_OPEN** and should be set to zero.

open

The union member of **LUA_SPECIFIC** used by **SLI_OPEN**. A supplied set of parameters contained in an **SLI_OPEN** structure required with **SLI_OPEN**.

open.lua_init_type

Supplied parameter. Defines how the LU-LU session is initialized by the Windows LUA interface.

Valid values are as follows:

LUA_INIT_TYPE_SEC_IS

LUA_INIT_TYPE_SEC_LOG

LUA_INIT_TYPE_PRIM

LUA_INIT_TYPE_PRIM_SSCP

open.lua_resv65

Reserved field.

open.lua_wait

Supplied parameter. Represents a secondary retry wait time indicating the number of seconds the Windows LUA interface is to wait before retrying the transmission of the INITSELF or the LOGON message after the host sends any one of these messages:

- A negative response and the secondary return code is one of the following:

  RESOURCE_NOT_AVAILABLE (0x08010000)SESSION_LIMIT_EXCEEDED (0x08050000)
  SESSION_SERVICE_PATH_ERROR (0x087D0000)

  Note that SLI_OPEN terminates with an error if lua_wait is set to zero and one of the preceding occurs.

- A network services procedure error (NSPE) message.

- A NOTIFY command, which indicates a procedure error.

open.lua_open_extension

Supplied parameter. Contains a list of application-supplied extension DLLs to process the BIND, STSN, and CRV commands.

open.open_extension.lua_routine_type

The extension routine type. Legal values are:

LUA_ROUTINE_TYPE_BIND

LUA_ROUTINE_TYPE_CRV

LUA_ROUTINE_TYPE_END (indicates end of extension list)

LUA_ROUTINE_TYPE_STSN

open.open_extension.lua_module_name

Supplied parameter. Provides the ASCII module name for the user-supplied extension DLL. The module name can be up to eight characters long, with the remaining bytes set to 0x00.

open.open_extension.lua_procedure_name

Supplied parameter. Provides the procedure name in ASCII for the user-supplied extension DLL. The procedure name can be up to 32 characters long, with the remaining bytes set to 0x00.

open.lua_ending_delim

The extension list delimiter.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** name was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_DATA_SEGMENT_LENGTH_ERROR

Secondary return code; one of the following occurred:

- The supplied data segment for SLI_RECEIVE or SLI_SEND is not a read/write data segment as required.

- The supplied data segment for SLI_RECEIVE is not as long as that provided in lua_max_length.
- The supplied data segment for SLI_SEND is not as long as that provided in lua_data_length.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_INVALID_OPEN_INIT_TYPE

Secondary return code; the value in the **lua_init_type** contained in **SLI_OPEN** is invalid.

LUA_INVALID_OPEN_DATA

Secondary return code; the **lua_init_type** for the **SLI_OPEN** issued is set to LUA_INIT_TYPE_SEC_IS when the buffer for data does not have a valid INITSELF command.

LUA_INVALID_OPEN_ROUTINE_TYPE

Secondary return code; the **lua_open_routine_type** for the **SLI_OPEN** list of extension routines is invalid.

LUA_DATA_LENGTH_ERROR

Secondary return code; the application did not provide user-supplied data required by the verb issued. Note that when SLI_SEND is issued for an SNA LUSTAT command, status (in four bytes) is required, and that when **SLI_OPEN** is issued with secondary initialization, data is required.

LUA_INVALID_SLI_ENCR_OPTION

Secondary return code; the **lua_encr_decr_option** parameter was set to 128 in **SLI_OPEN**, which is not supported for the encryption/decryption processing option.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_UNEXPECTED_SNA_SEQUENCE

Secondary return code; unexpected data or commands were received from the host while **SLI_OPEN** was processing.

LUA_NEG_RSP_FROM_BIND_ROUTINE

Secondary return code; the user-supplied SLI_BIND routine responded negatively to the BIND. **SLI_OPEN** ended unsuccessfully.

LUA_NEG_RSP_FROM_STSN_ROUTINE

Secondary return code; the user-supplied SLI STSN routine responded negatively to the STSN. **SLI_OPEN** ended unsuccessfully.

LUA_PROCEDURE_ERROR

Secondary return code; a host procedure error is indicated by the receipt of an NSPE or NOTIFY message. The return code is posted to **SLI_OPEN** when the retry option is not used. To use the reset option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue SLI_CLOSE.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued, or some verb other than **SLI_OPEN** was issued before the session was initialized.

LUA_RESOURCE_NOT_AVAILABLE

Secondary return code; the logical unit, physical unit, link, or link station specified in the request unit is unavailable. This return code is posted to **SLI_OPEN** when a resource is unavailable unless you use the retry option.

To use the retry option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue SLI_CLOSE.

LUA_SESSION_LIMIT_EXCEEDED

Secondary return code; the session requested was not activated because an NAU is at its session limit. This SNA sense code applies to the following requests: BID, CINIT, INIT, and ACTDRM.

The code will be posted to **SLI_OPEN** when an NAU is at its limit, unless you use the RETRY option.

To use the reset option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue **SLI_CLOSE**.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_NEGOTIABLE_BIND_ERROR

Secondary return code; a negotiable BIND was received, which is only allowed by the SLI when a user-supplied SLI_BIND routine is provided with **SLI_OPEN**.

LUA_BIND_FM_PROFILE_ERROR

Secondary return code; only file management header profiles 3 and 4 are supported by the LUA interface. A file management profile other than 3 or 4 was found on the BIND.

LUA_BIND_TS_PROFILE_ERROR

Secondary return code; only Transmission Service (TS) profiles 3 and 4 are supported by the LUA interface. A TS profile other than 3 or 4 was found on the BIND.

LUA_BIND_LU_TYPE_ERROR

Secondary return code; only LU 0, LU 1, LU 2, and LU 3 are supported by LUA. An LU other than 0, 1, 2, or 3 was found.

LUA_SSCP_LU_SESSION_NOT_ACTIVE

Secondary return code; the required SSCP-LU is inactive. Specific sense code information is in bytes 2 and 3. Valid settings are 0x0000, 0x0001, 0x0002, 0x0003, and 0x0004.

LUA_SESSION_SERVICES_PATH_ERROR

Secondary return code; a request for session services cannot be rerouted to an SSCP-SSCP session path. Specific sense code information in bytes 2 and 3 gives more information about why the request cannot be rerouted.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid but the verb did not complete successfully.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_SESSION_ALREADY_OPEN

Secondary return code; a session is already open for the LU name specified in **SLI_OPEN**.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another process owns the session.

LUA_LINK_NOT_STARTED

Secondary return code; the LUA was not able to activate the data link during initialization of the session.

LUA_INVALID_ADAPTER

Secondary return code; the configuration for the data link control (DLC) is in error, or the configuration file is corrupted.

LUA_ENCR_DECR_LOAD_ERROR

Secondary return code; an unexpected return code was received from the OS/2 **DosLoadModule** function while attempting to load the user-provided encryption or decryption dynamic link module.

LUA_ENCR_DECR_PROC_ERROR

Secondary return code; an unexpected return code was received from the OS/2 **DosGetProcAddr** function while attempting to get the procedure address within the user-provided encryption or decryption dynamic link module.

LUA_NEG_NOTIFY_RSP

Secondary return code; the SSCP responded negatively to a NOTIFY request issued indicating that the secondary LU was capable of a session. The half-session component that received the request understood and supported the request but could not execute it.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the SLI was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process was canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your

application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

For each **SLI_OPEN**, the Windows LUA interface:

- Starts the communication session.
- Reads and verifies a BIND command from the host, and passes it to the application if a BIND extension routine is supplied.
- Writes a BIND response.
- Reads and processes the STSN command and passes it to the application if a BIND extension is supplied (if necessary).
- Writes the STSN response (if necessary).
- Reads the CRV command (if necessary).
- Writes the CRV response (if necessary).
- Reads and processes the SDT command.
- Writes the SDT response.

The Windows LUA interface does the following additional functions for sessions that issue **SLI_OPEN** with the open type set to LUA_INIT_TYPE_SEC_IS or LUA_INIT_TYPE_SEC_LOG:

- Writes an INITSELF or an unformatted LOGON message.
- Reads and processes an INITSELF response or LOGON message response.

All SNA message traffic is administered by **SLI_OPEN** through the SDT command response.

To choose a certain LU configured for Windows LUA, the application sets lua_luname to the LU name in ASCII, padded with trailing spaces if necessary.

When SLI_OPEN is posted with LUA_OK in the lua_prim_rc parameter, SLI_OPEN successfully completed and the LU-LU data-flow session was established. The application can now issue SLI_BID, SLI_CLOSE, SLI_PURGE, SLI_RECEIVE, and SLI_SEND.

When SLI_OPEN is posted with a primary return code other than LUA_OK or LUA_IN_PROGRESS, the command did not successfully establish a session.

When using SLI_OPEN, a Windows LUA application must provide a session initialization type. Valid types are:

- Secondary with INITSELF
- Secondary with an Unformatted LOGON Message
- Primary Waiting for a BIND Command
- Primary with SSCP Access
- Secondary with INITSELF
- Secondary with an Unformatted LOGON Message

- Primary Waiting for a BIND Command
- Primary with SSCP Access
- BIND, CRV, and STSN Routines
- BIND Example
- Recovering from SESSION_FAILURE
- Ending a Pending SLI_OPEN

See Also

Reference

RUI_INIT
SLI_OPEN
SLI_RECEIVE
SLI_SEND

Secondary with INITSELF

To initialize a session by having the secondary issue an INITSELF command, set **open.lua_init_type** to LUA_INIT_TYPE_SEC_IS. When this type of session initialization is chosen, the application has to format and provide the INITSELF command. The address of the INITSELF command is specified by **lua_data_ptr**. The actual length of the INITSELF command is specified by **lua_data_length**.

Secondary with an Unformatted LOGON Message

To initialize a session by having the secondary issue an unformatted LOGON message, set **open.lua_init_type** to LUA_INIT_TYPE_SEC_LOG. The length of the users EBCDIC LOGON message is then specified in **lua_data_length**. The address of the users EBCDIC LOGON message length is specified by **lua_data_ptr**.

Primary Waiting for a BIND Command

To initialize a session by having the secondary wait for the primary to issue a BIND and SDT, set **open.lua_init_type** to LUA_INIT_TYPE_PRIM. Until the host begins a session with the Windows logical unit application (LUA) application using the BIND command followed by an SDT command, the **SLI_OPEN** issued stays IN_PROGRESS.

Primary with SSCP Access

To initialize a session by having the Session Level Interface (SLI) wait for a BIND and SDT but allow system services control point (SSCP) access, set **open.lua_init_type** to LUA_INIT_TYPE_PRIM_SSCP. Rather than sending commands to the host to begin a session, the SLI enables the Windows logical unit application (LUA) application to issue SLI_SEND and SLI_RECEIVE for the SSCP normal flow only. This allows the INITSELF commands or LOGON messages and responses to be transmitted between the Windows LUA application and the host. The application can have more than one INITSELF and LOGON message. For this type of session only, other SLI verbs can be issued before SLI_OPEN completes. When issuing **SLI_SEND**, an application should not specify any flow flag unless the application is sending a response, as specified in the **lua_message_type** parameter of **SLI_OPEN**. To obtain the INIT_COMPLETE status, the application must first issue **SLI_OPEN**, and then issue either SLI_BID or **SLI_RECEIVE**. The INIT_COMPLETE status notifies the application that the **SLI_SEND** and **SLI_RECEIVE** verbs for SSCP normal flow data can be issued.

BIND, CRV, and STSN Routines

For BIND and STSN routines supplied by the application, the names of dynamic-link libraries (DLLs) and the entry points for procedures are passed in the SLI_OPEN verbs verb control block (VCB). During **SLI_OPEN**, the BIND and STSN routines are called if the appropriate SNA request is received. When a BIND routine is not supplied by the application, the Session Level Interface (SLI) performs a minimal check of the BIND commands and responds as necessary. If no STSN routine is supplied and an STSN request arrives, a positive response is issued by the SLI. If a CRV request arrives, a negative response is issued by the SLI.

Names for BIND and STSN routines are provided as extensions of the SLI_OPEN verbs VCB. The lua_extension_list_offset parameter provides the offset from the start of the VCB to the first name in the extension list.

The function prototype for a user-defined BIND or STSN routine on Microsoft Windows Server 2003 or Windows 2000 is as follows:

Syntax

```
lpVcb
```

Remarks

The *lpVcb* parameter is a pointer to a logical unit application (LUA) VCB.

BIND Example

The following example illustrates checking the incoming BIND image using these features of SLI_OPEN.

```
lua_vcb.specific.open.lua_open_extension[0].lua_routine_type  =
    LUA_ROUTINE_TYPE_BIND;
strcpy(lua_vcb.specific.open.lua_open_extension[0].lua_module_name,
    "WINSLI32");
strcpy(lua_vcb.specific.open.lua_open_extension[0].lua_procedure_name,
    "BindValidation");
lua_vcb.specific.open.lua_open_extension[1].lua_routine_type =
     LUA_ROUTINE_TYPE_END;
```

Note that for Microsoft Visual C++ 4.0 or later, and for Microsoft Windows Server 2003 or Windows 2000, the function prototype should be:

```
VOID WINAPI BindValidation (LUA_VERB_RECORD FAR * pVerb );
```

On Windows Server 2003 or Windows 2000, the WINAPI macro equates to _STDCall.

The BIND routine has access to the logical unit application (LUA) verb control block (VCB) passed to it. The BIND routine should validate the BIND and indicate the appropriate Session Level Interface (SLI) primary and secondary return code in the LUA verb record. Also, the routine may indicate the primary and secondary request/response unit (RU) sizes supported by the SLI program by setting bytes 10 and 11 in the common.lua_data_ptr field (where the BIND command is indicated).

The following are the Visual C++ compiler options for the module containing the callback:

```
 /FA -c -Zle -W3 -WX -Ge -Gy -Gz -Ox -Zd
 -DCONDITION_HANDLING -DSTD_CALL
 -Di386=1 -D_X86_ -DNT_UP -DWIN32 -DDEVL
 -D_DLL -D_MT  -DWIN32_SUPPORT
```

The following is the code generated for the callback:

```
PUBLIC _BindValidation@4
; COMDAT _BindValidation@4
_TEXT SEGMENT
   _pVerb$ = 8
   _BindValidation@4 PROC NEAR    ; COMDAT

   // pVerb->common.lua_prim_rc = LUA_STATE_CHECK;
   mov eax, DWORD PTR _pVerb$[esp-4]
   mov WORD PTR [eax+4], 512   ; 00000200H
   ret 4
_BindValidation@4 ENDP
_TEXT ENDS
```

The following is the code generated by SLI to call this callback:

```
   // (*aSCB->bind_rtn)(sliVCB);
   push    ebp
   call    DWORD PTR [ebx+188]
   // note there is no ADD ESP,4 following the call
```

The following is the client internal trace showing WINSLI detecting the user provided bind validation callback:

```
|00000157.000000f7 OUDMD  Opening User DLL Modules
```

```
|00000157.000000f7 OUDMD  Opening a Bind Routine
|00000157.000000f7 OUDMD  Opening DLL = WINSLI32
|00000157.000000f7 OUDMD  Loading Routine = BindValidation
```

The following is client internal trace showing the bind validation callback:

```
|00000157.0000015c CLUAD  Calling BIND Routine
|00000157.0000015c CLUAD  Return from BIND routine, prc=512
|00000157.0000015c CLUAD  Returned With Error From Routine
|00000157.0000015c FrRUI  Freeing RUI vcb = 0x14E424
|00000157.0000015c BINDP  USER BIND ROUTINE FAILED
```

The following is an API trace to show the bind validation error:

```
000015c SLI    ------------------------------------------   11:11:52.28
000015c SLI    SLI_OPEN post
000015c SLI    SESSION_FAILURE - NEG_RSP_FROM_BIND_ROUTINE
000015c SLI    ---- Verb Parameter Block at address 00405150 ----
000015c SLI    52004900 000F0000 00000039 01000000
    <R.I........9....>
000015c SLI    00000000 4C553220 20202020 48000000
    <....LU2     H...>
000015c SLI    88E01400 00000400 C0904000 F4000000
    <h.........@.4...>
000015c SLI    00000000 00000000 00000040 00000000
    <...........@....>
000015c SLI    00000000 02000000 0157494E 534C4933
    <.........WINSLI3>
000015c SLI    32004269 6E645661 6C696461 74696F6E
    <2.BindValidation>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 00000000 00000000
    <................>
000015c SLI    00000000 00000000 0000
    <..........      >
000015c SLI    ---- Data at address 004090C0 ----
000015c SLI    86998584
    <fred            >
```

Recovering from SESSION_FAILURE

If the SLI_OPEN completes with the primary return code of SESSION_FAILURE, the Windows logical unit application (LUA) interface enables you to reissue **SLI_OPEN** without issuing SLI_CLOSE.

Ending a Pending SLI_OPEN

To end a pending SLI_OPEN, issue SLI_CLOSE with **lua_flag2.close_abend** set to ON.

SLI_PURGE

The **SLI_PURGE** verb cancels SLI_RECEIVE verbs issued with a wait condition.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_PURGE.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the logical unit application (LUA) VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_PURGE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_PURGE only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_PURGE** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_PURGE** and should be set to zero.

lua_data_length

Not used by **SLI_PURGE** and should be set to zero.

lua_data_ptr

When **SLI_PURGE** is issued, this parameter points to the location of the SLI_RECEIVE verbs VCB that is to be canceled.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows® 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Not used by **SLI_PURGE** and should be set to zero.

lua_rh

Not used by **SLI_PURGE** and should be set to zero.

lua_flag1

Not used by **SLI_PURGE** and should be set to zero.

lua_message_type

Not used by **SLI_PURGE** and should be set to zero.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_PURGE** and should be set to zero.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for the Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_NO_RECEIVE_TO_PURGE

Secondary return code; no SLI_RECEIVE was outstanding when you issued **SLI_PURGE**. One of two situations caused the problem:

- **SLI_RECEIVE** completed before **SLI_PURGE** finished processing. You can change the application to take care of this problem because it is not an error condition.
- The lua_data_ptr parameter does not correctly point to the SLI_RECEIVE you want to purge.

LUA_SLI_PURGE_PENDING

Secondary return code; an **SLI_PURGE** was still active when another **SLI_PURGE** was issued. Only one **SLI_PURGE** can be active at a time.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid but the verb did not complete successfully.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_NOT_READY

Secondary return code; one of the following caused the SLI session to be temporarily suspended:

- An SNA UNBIND type 0x02 command was received, which indicates a new BIND is coming. If the UNBIND type 0x02 is received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while the **SLI_OPEN** is still processing, the primary return code is SESSION_FAILURE, not LUA_STATUS.

- The receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_INVALID_PROCESS

Secondary return code; the session for which a Request Unit Interface (RUI) verb was issued is unavailable because another OS/2 process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the RUI was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process was canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.

- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).

- The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_PURGE cancels SLI_RECEIVE commands with a wait condition.

Typically, SLI_PURGE is issued if SLI_RECEIVE takes too long to complete. To cancel an SLI_RECEIVE, lua_data_ptr has to point to the SLI_RECEIVE VCB to cancel. The primary return code of the SLI_RECEIVE will be set to LUA_CANCELED when SLI_PURGE succeeds in canceling SLI_RECEIVE.

See Also

Reference

RUI_INIT
SLI_OPEN
SLI_PURGE
SLI_RECEIVE
SLI_SEND

SLI_RECEIVE

The **SLI_RECEIVE** verb receives responses, SNA commands, and data into a Microsoft® Windows® logical unit application (LUA) applications buffer. **SLI_RECEIVE** also provides the current status of the session to the Windows LUA application.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_RECEIVE.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_RECEIVE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_RECEIVE only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_RECEIVE** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Specifies the length of received buffer for RUI_READand **SLI_RECEIVE**.

lua_data_length

Returned parameter. Specifies the length of data returned in the receive buffer.

lua_data_ptr

Pointer to the application-supplied buffer that is to receive the data from an **SLI_RECEIVE** verb. Both SNA commands and data are placed in this buffer, and they can be in an Extended Binary Coded Decimal Interchange Code (EBCDIC) format.

When SLI_RECEIVE is issued, this parameter points to the location to receive the data from the host.

lua_post_handle

Supplied parameter. Used under Microsoft® Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x0 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

Request/response unit (RU) category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. This parameter is used by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, **SLI_RECEIVE**, and SLI_SEND. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set **lua_flag1.bid_enable** to 1 to re-enable the most recent SLI_BID (equivalent to issuing **SLI_BID** again with exactly the same parameters as before), or set it to zero if you do not want to re-enable **SLI_BID**. Note that re-enabling the previous **SLI_BID** reuses the VCB originally allocated for it, so this VCB must not have been freed or modified.

Set lua_flag1.nowait to 1 to indicate that you want SLI_RECEIVE to return immediately whether or not data is available to be read, or set it to zero if you want the verb to wait for data before returning.

Set one or more of the following flags to 1 to indicate from which message flow to read data:

lua_flag1.sscp_exp

lua_flag1.lu_exp

lua_flag1.sscp_norm

lua_flag1.lu_norm

If more than one flag is set, the highest-priority data available is returned. The order of priorities (highest first) is: SSCP expedited, LU expedited, SSCP normal, LU normal. The equivalent flag in the **lua_flag2** group is set to indicate from which flow the data was read.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. Returned parameter. Specifies the type of SNA message indicated to **SLI_RECEIVE**. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_STSN

The SLI receives and responds to the BIND and STSN requests through the LUA interface extension routines.

LU-DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Returned by RUI_BID, RUI_READ, **RUI_WRITE**, **SLI_BID**, **SLI_RECEIVE**, and SLI_SEND. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that **RUI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Not used by **SLI_RECEIVE** and should be set to zero.

lua_encr_decr_option

Not used by **SLI_RECEIVE** and should be set to zero.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous

posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_BID_VERB_SEGMENT_ERROR

Secondary return code; the buffer with the SLI_BID VCB was released before the **SLI_RECEIVE** with **lua_flag1.bid_enable** set to 1 was issued.

LUA_NO_PREVIOUS_BID_ENABLED

Secondary return code; **SLI_BID** was not issued prior to issuing **SLI_RECEIVE** with **lua_flag1.bid_enable**.

LUA_BID_ALREADY_ENABLED

Secondary return code; **SLI_RECEIVE** was issued with **lua_flag1.bid_enable** when **SLI_BID** was already active.

LUA_INVALID_FLOW

Secondary return code; the **lua_flag1** flow flags were set incorrectly when a verb was issued:

- When issuing SLI_SEND to send an SNA response, set only one **lua_flag1** flow flag.
- When issuing SLI_RECEIVE, set at least one lua_flag1 flow flag.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_RECEIVE_ON_FLOW_PENDING

Secondary return code; an **SLI_RECEIVE** was still outstanding when this application issued another **SLI_RECEIVE** for an SNA flow.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RUI_WRITE_FAILURE

Secondary return code; an unexpected error was posted to the SLI by RUI_WRITE.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_RECEIVER_IN_TRANSMIT_MODE

Secondary return code; either resources needed to handle normal flow data were not available or the state of the half-duplex contention was not received when a normal-flow request was received. The result is a race condition. This SNA sense code is

also an exception request sense code.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_CHAINING_ERROR

Secondary return code; the sequence of the chain indicator settings is in error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_BRACKET

Secondary return code; the sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DIRECTION

Secondary return code; while the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_QUIESCED

Secondary return code; a data flow control (DFC) or function management data (FMD) request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_NO_BEGIN_BRACKET

Secondary return code; the receiver has already sent a positive response to a BIS command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_IMMEDIATE_REQUEST_MODE_ERROR

Secondary return code; the request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_QUEUED_RESPONSE_ERROR

Secondary return code; the request violated the queued response protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_ERP_SYNC_EVENT_ERROR

Secondary return code; a violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_CORRELATION_ERROR

Secondary return code; a response was sent that does not correspond to a previously received request or a response was received that does not correspond to a previously sent request.

LUA_RSP_PROTOCOL_ERROR

Secondary return code; a violation of the response protocol was found in the response received from the primary half-session.

LUA_BB_NOT_ALLOWED

Secondary return code; the begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was

prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EB_NOT ALLOWED

Secondary return code; the end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EXCEPTION_RSP_NOT_ALLOWED

Secondary return code; when an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_DEFINITE_RSP_NOT_ALLOWED

Secondary return code; when a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_ALLOWED

Secondary return code; the change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_RESPONSE_NOT_ALLOWED

Secondary return code; a request other than an EXR contained a NO RESPONSE. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CHAINING_NOT_SUPPORTED

Secondary return code; the chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_BRACKETS_NOT_SUPPORTED

Secondary return code; the bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_SUPPORTED

Secondary return code; the change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_FI

Secondary return code; the format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_ALTERNATE_CODE_NOT_SUPPORTED

Secondary return code; the code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to

the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_RU_CATEGORY

Secondary return code; the request unit category indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_REQUEST_CODE

Secondary return code; the request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_SPEC_OF_SDI_RTI

Secondary return code; the SDI and the RTI were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_DR1I_DR2I_ERI

Secondary return code; the DR1I, the DR2I, and the ERI were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_QRI

Secondary return code; the queued response indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_EDI

Secondary return code; the EDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_PDI

Secondary return code; the PDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid but the verb did not complete successfully.

LUA_DATA_TRUNCATED

Secondary return code; the data was truncated because the data received was longer than the buffer length specified in **lua_max_length**.

LUA_DATA_SEGMENT_LENGTH_ERROR

Secondary return code; one of the following has occurred:

- The supplied data segment for **SLI_RECEIVE** or SLI_SEND is not a read/write data segment as required.
- The supplied data segment for SLI_RECEIVE is not as long as that provided in lua_max_length.
- The supplied data segment for SLI_SEND is not as long as that provided in lua_data_length.

LUA_NO_DATA

Secondary return code; no data was available to read when **SLI_RECEIVE** containing a no wait parameter was issued.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_NOT_READY

Secondary return code; one of the following has caused the SLI session to be temporarily suspended:

- An SNA UNBIND type 0x02 command was received, which indicates a new BIND is coming. If the UNBIND type 0x02 is received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while the **SLI_OPEN** is still processing, the primary return code is LUA_SESSION_FAILURE, not LUA_STATUS.

- The receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another OS/2 process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the LUA was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_RECEIVE_CORRELATION_TABLE_FULL

Secondary return code; the session receive correlation table for the flow requested reached its capacity.

LUA_NEGATIVE_RESPONSE

Primary return code; either the LUA sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; the LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_DATA_TRAFFIC_RESET

Secondary return code; a half-session of an active session but with inactive data traffic received a normal flow DFC or FMD request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_NOT_RESET

Secondary return code; while the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_SC_PROTOCOL_VIOLATION

Secondary return code; a violation of SC protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_INVALID_SC_OR_NC_RH

Secondary return code; the RH of an SC or NC request was invalid.

LUA_PACING_NOT_SUPPORTED

Secondary return code; the request contained a pacing indicator when support of pacing for this session does not exist for the receiving half-session or boundary function half-session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NAU_INOPERATIVE

Secondary return code; the network addressable unit is not able to process responses or requests. Delivery to the receiver could not take place for one of the following reasons:

- A path information unit error
- A path outage
- An invalid sequence of requests for activation

If a path error is received during an active session, it usually means there is no longer a valid path to the session partner.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_PURGED

Secondary return code; SLI_PURGE was issued and canceled **SLI_RECEIVE**.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_CANCEL_COMMAND_RECEIVED

Secondary return code; the host sent an SNA CANCEL command to cancel the data chain currently being received by **SLI_RECEIVE**.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process has been canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_STATUS

Primary return code; the secondary return code contains SLI status information for the application.

LUA_READY

Secondary return code; following a NOT READY status, this status is issued to notify you that the SLI is ready to process commands.

LUA_NOT_READY

Secondary return code; the SLI session is temporarily suspended for the following reason:

- An SNA UNBIND type 0x02 command was received, which means a new BIND is coming. If the UNBIND type 0x02 is

received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while the **SLI_OPEN** is still processing, the primary return code is session-failure, not status.

- The receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_INIT_COMPLETE

Secondary return code; the LUA interface initialized the session while SLI_OPEN was processing. LUA applications that issue **SLI_OPEN** with **lua_open_type_prim_sscp** receive this status on **SLI_RECEIVE** or SLI_BID.

LUA_SESSION_END_REQUESTED

Secondary return code; the LUA interface received an SNA SHUTD from the host, which means the host is ready to shut down the session.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_RECEIVE receives responses, SNA commands, and request unit data from the host. **SLI_RECEIVE** also provides the status of the session to the Windows LUA application. An SLI_OPEN request must complete before **SLI_RECEIVE** can be issued. However, if **SLI_OPEN** is issued with **lua_init_type** set to LUA_INIT_TYPE_PRIM_SSCP, an **SLI_RECEIVE** over the SSCP normal flow can be issued as soon as **SLI_OPEN** returns an IN_PROGRESS.

Data is received by the application in one of four session flows. The four session flows, from highest to lowest priority are:

- SSCP expedited
- LU expedited
- SSCP normal

- LU normal

The data flow type that **SLI_RECEIVE** will process is specified in **lua_flag1**. The application can also specify whether it wants to look at more than one type of data flow. When multiple flow bits are set, the highest priority is received first. When **SLI_RECEIVE** completes processing, **lua_flag2** indicates the specific type of flow for which data has been received by the Windows LUA application.

If SLI_BID successfully completes before SLI_RECEIVE is issued, the Windows LUA interface can be instructed to reuse the last SLI_BID verbs VCB. To do this, issue SLI_RECEIVE with lua_flag1.bid_enable set to 1.

When using lua_flag1.bid_enable, the SLI_BID storage must not be freed because the last SLI_BID verbs VCB is used. Also, when using lua_flag1.bid_enable, the successful completion of SLI_BID will be posted.

If SLI_RECEIVE is issued with lua_flag1.nowait when no data is available to receive, LUA_NO_DATA will be the secondary return code set by the Windows LUA interface.

Session Status Return Values

If LUA_STATUS is the primary return code, the secondary return code can be one of the following:

LUA_READY

LUA_NOT_READY

LUA_SESSION_END_REQUESTED

LUA_INIT_COMPLETE

In addition, if LUA_STATUS is the primary return code, the following parameters are used:

lua_sec_rc

lua_sid

LUA_READY is returned after an LUA_NOT_READY status and indicates that the SLI is again ready to perform all commands.

LUA_NOT_READY indicates that the SLI session is suspended because the SLI has received either an SNA CLEAR command or an SNA UNBIND command with an 0x02 UNBIND type (UNBIND with BIND forthcoming). Depending on what caused the suspension, the session can be reactivated as follows:

- When the suspension is caused by an SNA CLEAR, receiving an SNA SDT reactivates the session.

- When an SNA UNBIND type BIND forthcoming causes suspension of the session and the SLI_OPEN that opened the session is completed, the session is suspended until the SLI receives a BIND and SDT command. The session can also optionally receive an STSN command. As a result, user-supplied routines issued with the initial SLI_OPEN must be re-entered because they will be recalled.

The application can send SSCP data after a CLEAR or UNBIND type BIND forthcoming arrives and before the NOT_READY status is read. The application can send and receive SSCP data after reading a NOT_READY.

When an SNA UNBIND type BIND forthcoming arrives before completion of the SLI_OPEN that opened the session, LUA_SESSION_FAILURE (not LUA_STATUS) is the primary return code.

LUA_SESSION_END_REQUESTED indicates that the application received an SNA SHUTD from the host. The Windows LUA application should issue SLI_CLOSE to close the session when convenient.

LUA_INIT_COMPLETE is returned only when lua_init_type for SLI_OPEN is LUA_INIT_TYPE_PRIM_SSCP. The status means that the SLI_OPEN has been processed sufficiently to allow SSCP data to now be sent or received.

Exception Requests

If a host application request unit is converted into an EXR, sense data will be returned. When SLI_BID completes with the returned verb parameters set as shown, an EXR conversion occurs.

Member	Set to
lua_prim_rc	OK (0x0000)

lua_sec_rc	OK (0x00000000)
lua_rh.rri	bit off (request unit)
lua_rh.sdi	bit on (includes sense data)

Of the seven bytes of data in **lua_peek_data**, bytes 0 through 3 define the error detected. The following table indicates possible sense data and the values of bytes 0 through 3.

Sense data	Value of bytes 0–3
LUA_MODE_INCONSISTENCY	0x08090000
LUA_BRACKET_RACE_ERROR	0x080B0000
LUA_BB_REJECT_NO_RTR	0x08130000
LUA_RECEIVER_IN_TRANSMIT_MODE	0x081B0000
LUA_CRYPTOGRAPHY_FUNCTION_INOP	0x08480000
LUA_SYNC_EVENT_RESPONSE	0x10010000
LUA_RU_DATA_ERROR	0x10020000
LUA_RU_LENGTH_ERROR	0x10020000
LUA_INCORRECT_SEQUENCE_NUMBER	0x20010000

The information returned to bytes 3 through 6 in **lua_peek_data** is determined by the first three bytes of the initial request unit that caused the error.

See Also

Reference

RUI_INIT
RUI_PURGE
RUI_READ
RUI_WRITE
SLI_BID
SLI_CLOSE
SLI_OPEN
SLI_PURGE
SLI_SEND

SLI_RECEIVE_EX

The **SLI_RECEIVE_EX** verb receives responses, SNA commands, and data into a Microsoft® Windows® logical unit application (LUA) applications buffer. **SLI_RECEIVE_EX** also provides the current status of the session to the Windows LUA application.

The SLI_RECEIVE_EX verb also supports inbound chaining. The maximum length of data that can be received by a single verb is 4,294,967,295 bytes. This is compared to a maximum of 65,535 bytes that can be received by the SLI_RECEIVE verb.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_RECEIVE_EX.

The second syntax union describes the **LUA_SPECIFIC** member of the VCB used by **SLI_RECEIVE_EX**. Other union members are omitted for clarity.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
union LUA_SPECIFIC {
    struct SLI_RECEIVE_EX_SPECIFIC {
        unsigned long lua_data_length_ex;
        unsigned long lua_max_length_ex;
    };
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_RECEIVE_EX.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_RECEIVE_EX only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_RECEIVE_EX** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

This supplied parameter is reserved and must be set to zero.

The maximum length of data returned in a receive buffer must be set in the lua_max_length_ex parameter.

lua_data_length

This parameter is reserved and must be set to zero.

The length of data returned in the receive buffer is set in the lua_data_length_ex parameter.

lua_data_ptr

Pointer to the application-supplied buffer that is to receive the data from an **SLI_RECEIVE_EX** verb. Both SNA commands and data are placed in this buffer, and they can be in an Extended Binary Coded Decimal Interchange Code (EBCDIC) format.

When SLI_RECEIVE_EX is issued, this parameter points to the location to receive the data from the host.

lua_post_handle

Supplied parameter. Used under Microsoft® Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. Its subparameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. This parameter is used by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, **SLI_RECEIVE_EX**, and SLI_SEND_EX. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set **lua_flag1.bid_enable** to 1 to re-enable the most recent SLI_BID (equivalent to issuing **SLI_BID** again with exactly the same parameters as before), or set it to zero if you do not want to re-enable **SLI_BID**. Note that re-enabling the previous **SLI_BID** reuses the VCB originally allocated for it, so this VCB must not have been freed or modified.

Set lua_flag1.nowait to 1 to indicate that you want SLI_RECEIVE_EX to return immediately whether or not data is available to be read, or set it to zero if you want the verb to wait for data before returning.

Set one or more of the following flags to 1 to indicate from which message flow to read data:

lua_flag1.sscp_exp

lua_flag1.lu_exp

lua_flag1.sscp_norm

lua_flag1.lu_norm

If more than one flag is set, the highest-priority data available is returned. The order of priorities (highest first) is: SSCP expedited, LU expedited, SSCP normal, LU normal. The equivalent flag in the **lua_flag2** group is set to indicate from which flow the data was read.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. Returned parameter. Specifies the type of SNA message indicated to **SLI_RECEIVE_EX**. Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_STSN

The SLI receives and responds to the BIND and STSN requests through the LUA interface extension routines.

LU-DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Returned by RUI_BID, RUI_READ, **RUI_WRITE**, **SLI_BID**, **SLI_RECEIVE**, and SLI_SEND_EX. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that **RUI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Not used by **SLI_RECEIVE** and should be set to zero.

lua_encr_decr_option

Not used by **SLI_RECEIVE** and should be set to zero.

lua_max_length_ex

Specifies the length of received buffer for **SLI_RECEIVE_EX**.

lua_data_length_ex

The union member of **LUA_SPECIFIC** used by **SLI_RECEIVE_EX**. Returned parameter. Specifies the length of data returned in the receive buffer.

Return Codes
LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_BID_VERB_SEGMENT_ERROR

Secondary return code; the buffer with the SLI_BID VCB was released before the **SLI_RECEIVE_EX** with **lua_flag1.bid_enable** set to 1 was issued.

LUA_NO_PREVIOUS_BID_ENABLED

Secondary return code; **SLI_BID** was not issued prior to issuing **SLI_RECEIVE_EX** with **lua_flag1.bid_enable**.

LUA_BID_ALREADY_ENABLED

Secondary return code; **SLI_RECEIVE_EX** was issued with **lua_flag1.bid_enable** when **SLI_BID** was already active.

LUA_INVALID_FLOW

Secondary return code; the **lua_flag1** flow flags were set incorrectly when a verb was issued:

When issuing SLI_SEND_EX_sna_SLI_SEND_EX_lua to send an SNA response, set only one lua_flag1 flow flag.

When issuing SLI_RECEIVE, set at least one lua_flag1 flow flag.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_RECEIVE_ON_FLOW_PENDING

Secondary return code; an **SLI_RECEIVE_EX** was still outstanding when this application issued another **SLI_RECEIVE_EX** for an SNA flow.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RUI_WRITE_FAILURE

Secondary return code; an unexpected error was posted to the SLI by RUI_WRITE.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_RECEIVER_IN_TRANSMIT_MODE

Secondary return code; either resources needed to handle normal flow data were not available or the state of the half-duplex contention was not received when a normal-flow request was received. The result is a race condition. This SNA sense code is also an exception request sense code.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; LUA does not support the requested function. A control character, a request/response unit (RU) parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_CHAINING_ERROR

Secondary return code; the sequence of the chain indicator settings is in error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_BRACKET

Secondary return code; the sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DIRECTION

Secondary return code; while the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_QUIESCED

Secondary return code; a data flow control (DFC) or function management data (FMD) request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_NO_BEGIN_BRACKET

Secondary return code; the receiver has already sent a positive response to a BIS command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_IMMEDIATE_REQUEST_MODE_ERROR

Secondary return code; the request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_QUEUED_RESPONSE_ERROR

Secondary return code; the request violated the queued response protocol. An invalid header request or request unit for the

received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_ERP_SYNC_EVENT_ERROR

Secondary return code; a violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_CORRELATION_ERROR

Secondary return code; a response was sent that does not correspond to a previously received request or a response was received that does not correspond to a previously sent request.

LUA_RSP_PROTOCOL_ERROR

Secondary return code; a violation of the response protocol was found in the response received from the primary half-session.

LUA_BB_NOT_ALLOWED

Secondary return code; the begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EB_NOT ALLOWED

Secondary return code; the end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EXCEPTION_RSP_NOT_ALLOWED

Secondary return code; when an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_DEFINITE_RSP_NOT_ALLOWED

Secondary return code; when a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_ALLOWED

Secondary return code; the change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_RESPONSE_NOT_ALLOWED

Secondary return code; a request other than an EXR contained a NO RESPONSE. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CHAINING_NOT_SUPPORTED

Secondary return code; the chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_BRACKETS_NOT_SUPPORTED

Secondary return code; the bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was

prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_SUPPORTED

Secondary return code; the change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_FI

Secondary return code; the format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_ALTERNATE_CODE_NOT_SUPPORTED

Secondary return code; the code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_RU_CATEGORY

Secondary return code; the request unit category indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_REQUEST_CODE

Secondary return code; the request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_SPEC_OF_SDI_RTI

Secondary return code; the SDI and the RTI were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_DR1I_DR2I_ERI

Secondary return code; the DR1I, the DR2I, and the ERI were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_QRI

Secondary return code; the queued response indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_EDI

Secondary return code; the EDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_PDI

Secondary return code; the PDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_UNSUCCESSFUL

Primary return code; the verb record supplied was valid but the verb did not complete successfully.

LUA_DATA_TRUNCATED

Secondary return code; the data was truncated because the data received was longer than the buffer length specified in **lua_max_length_ex**.

LUA_DATA_SEGMENT_LENGTH_ERROR

Secondary return code; one of the following has occurred:

The supplied data segment for **SLI_RECEIVE_EX** or SLI_SEND_EX is not a read/write data segment as required.

The supplied data segment for SLI_RECEIVE_EX is not as long as that provided in lua_max_length_ex.

The supplied data segment for SLI_SEND_EX is not as long as that provided in lua_data_length_ex.

LUA_NO_DATA

Secondary return code; no data was available to read when **SLI_RECEIVE_EX** containing a no wait parameter was issued.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_NOT_READY

Secondary return code; one of the following has caused the SLI session to be temporarily suspended:

An SNA UNBIND type 0x02 command was received, which indicates a new BIND is coming. If the UNBIND type 0x02 is received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while the **SLI_OPEN** is still processing, the primary return code is LUA_SESSION_FAILURE, not LUA_STATUS.

The receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the LUA was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_RECEIVE_CORRELATION_TABLE_FULL

Secondary return code; the session receive correlation table for the flow requested reached its capacity.

LUA_NEGATIVE_RESPONSE

Primary return code; either the LUA sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; the LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_DATA_TRAFFIC_RESET

Secondary return code; a half-session of an active session but with inactive data traffic received a normal flow DFC or FMD request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_NOT_RESET

Secondary return code; while the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_SC_PROTOCOL_VIOLATION

Secondary return code; a violation of SC protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_INVALID_SC_OR_NC_RH

Secondary return code; the RH of an SC or NC request was invalid.

LUA_PACING_NOT_SUPPORTED

Secondary return code; the request contained a pacing indicator when support of pacing for this session does not exist for the receiving half-session or boundary function half-session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NAU_INOPERATIVE

Secondary return code; the network addressable unit is not able to process responses or requests. Delivery to the receiver could not take place for one of the following reasons:

A path information unit error

A path outage

An invalid sequence of requests for activation

If a path error is received during an active session, it usually means there is no longer a valid path to the session partner.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_PURGED

Secondary return code; SLI_PURGE was issued and canceled **SLI_RECEIVE**.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_CANCEL_COMMAND_RECEIVED

Secondary return code; the host sent an SNA CANCEL command to cancel the data chain currently being received by **SLI_RECEIVE_EX**.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process has been canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_STATUS

Primary return code; the secondary return code contains SLI status information for the application.

LUA_READY

Secondary return code; following a NOT READY status, this status is issued to notify you that the SLI is ready to process commands.

LUA_NOT_READY

Secondary return code; the SLI session is temporarily suspended for the following reason:

An SNA UNBIND type 0x02 command was received, which means a new BIND is coming. If the UNBIND type 0x02 is received after the beginning SLI_OPEN is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they have to be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while the **SLI_OPEN** is still processing, the primary return code is session-failure, not status.

The receipt of an SNA CLEAR caused the suspension. Receipt of an SNA SDT will cause the session to resume.

LUA_INIT_COMPLETE

Secondary return code; the LUA interface initialized the session while SLI_OPEN was processing. LUA applications that issue **SLI_OPEN** with **lua_open_type_prim_sscp** receive this status on **SLI_RECEIVE** or SLI_BID.

LUA_SESSION_END_REQUESTED

Secondary return code; the LUA interface received an SNA SHUTD from the host, which means the host is ready to shut down the session.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation encountered an ABEND.

The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).

The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_RECEIVE_EX receives responses, SNA commands, and request unit data from the host. **SLI_RECEIVE_EX** also provides the status of the session to the Windows LUA application.

The difference between SLI_RECEIVE_EX and SLI_RECEIVE is that the SLI_RECEIVE_EX verb supports inbound chaining and can receive up to 4,295 kilobytes (KB) in a single verb request. In contrast, SLI_RECEIVE is limited to receiving up to 64 KB in a verb request.

An SLI_OPEN request must complete before SLI_RECEIVE_EX can be issued. However, if SLI_OPEN is issued with lua_init_type set to LUA_INIT_TYPE_PRIM_SSCP, an SLI_RECEIVE_EX over the SSCP normal flow can be issued as soon as SLI_OPEN returns an IN_PROGRESS.

Data is received by the application in one of four session flows. The four session flows, from highest to lowest priority are:

- SSCP expedited
- LU expedited
- SSCP normal
- LU normal

The data flow type that **SLI_RECEIVE_EX** will process is specified in **lua_flag1**. The application can also specify whether it wants to look at more than one type of data flow. When multiple flow bits are set, the highest priority is received first. When **SLI_RECEIVE_EX** completes processing, **lua_flag2** indicates the specific type of flow for which data has been received by the Windows LUA application.

If SLI_BID successfully completes before SLI_RECEIVE is issued, the Windows LUA interface can be instructed to reuse the last SLI_BID verbs VCB. To do this, issue SLI_RECEIVE_EX with lua_flag1.bid_enable set to 1.

When using lua_flag1.bid_enable, the SLI_BID storage must not be freed because the last SLI_BID verbs VCB is used. Also, when using lua_flag1.bid_enable, the successful completion of SLI_BID will be posted.

If SLI_RECEIVE_EX is issued with lua_flag1.nowait when no data is available to receive, LUA_NO_DATA will be the secondary return code set by the Windows LUA interface.

Session Status Return Values

If LUA_STATUS is the primary return code, the secondary return code can be one of the following:

LUA_READY

LUA_NOT_READY

LUA_SESSION_END_REQUESTED

LUA_INIT_COMPLETE

In addition, if LUA_STATUS is the primary return code, the following parameters are used:

lua_sec_rc

lua_sid

LUA_READY is returned after an LUA_NOT_READY status and indicates that the SLI is again ready to perform all commands.

LUA_NOT_READY indicates that the SLI session is suspended because the SLI has received either an SNA CLEAR command or an SNA UNBIND command with an 0x02 UNBIND type (UNBIND with BIND forthcoming). Depending on what caused the suspension, the session can be reactivated as follows:

- When the suspension is caused by an SNA CLEAR, receiving an SNA SDT reactivates the session.
- When an SNA UNBIND type BIND forthcoming causes suspension of the session and the SLI_OPEN that opened the session is completed, the session is suspended until the SLI receives a BIND and SDT command. The session can also optionally receive an STSN command. As a result, user-supplied routines issued with the initial SLI_OPEN must be re-entered because they will be recalled.

The application can send SSCP data after a CLEAR or UNBIND type BIND forthcoming arrives and before the NOT_READY status is read. The application can send and receive SSCP data after reading a NOT_READY.

When an SNA UNBIND type BIND forthcoming arrives before completion of the SLI_OPEN that opened the session, LUA_SESSION_FAILURE (not LUA_STATUS) is the primary return code.

LUA_SESSION_END_REQUESTED indicates that the application received an SNA SHUTD from the host. The Windows LUA application should issue SLI_CLOSE to close the session when convenient.

LUA_INIT_COMPLETE is returned only when lua_init_type for SLI_OPEN is LUA_INIT_TYPE_PRIM_SSCP. The status means that the SLI_OPEN has been processed sufficiently to allow SSCP data to now be sent or received.

Exception Requests

If a host application request unit is converted into an EXR, sense data will be returned. When SLI_BID completes with the returned verb parameters set as shown, an EXR conversion occurs.

Member	Set to
lua_prim_rc	OK (0x0000)
lua_sec_rc	OK (0x00000000)
lua_rh.rri	bit off (request unit)
lua_rh.sdi	bit on (includes sense data)

Of the seven bytes of data in **lua_peek_data**, bytes 0 through 3 define the error detected. The following table indicates possible sense data and the values of bytes 0 through 3.

Sense data	Value of bytes 0–3
LUA_MODE_INCONSISTENCY	0x08090000
LUA_BRACKET_RACE_ERROR	0x080B0000
LUA_BB_REJECT_NO_RTR	0x08130000
LUA_RECEIVER_IN_TRANSMIT_MODE	0x081B0000
LUA_CRYPTOGRAPHY_FUNCTION_INOP	0x08480000
LUA_SYNC_EVENT_RESPONSE	0x10010000
LUA_RU_DATA_ERROR	0x10020000
LUA_RU_LENGTH_ERROR	0x10020000
LUA_INCORRECT_SEQUENCE_NUMBER	0x20010000

The information returned to bytes 3 through 6 in **lua_peek_data** is determined by the first three bytes of the initial request unit that caused the error.

See Also

Reference

RUI_INIT
RUI_PURGE
RUI_READ
RUI_WRITE
SLI_BID
SLI_CLOSE
SLI_OPEN
SLI_PURGE
SLI_SEND_EX

SLI_SEND

The **SLI_SEND** verb sends responses, SNA commands, and data from a Microsoft® Windows® logical unit application (LUA) application to a host logical unit (LU).

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_SEND.

The second syntax union below describes the **LUA_SPECIFIC** member of the VCB used by **SLI_SEND**. Other union members are omitted for clarity.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
union LUA_SPECIFIC {
    unsigned char lua_sequence_number[2];
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_SEND.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_SEND only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_SEND** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_SEND** and should be set to zero.

lua_data_length

Supplied parameter. Specifies the length of data being sent.

lua_data_ptr

Pointer to the application-supplied buffer that contains the data to be sent to the host by **SLI_SEND**.

Both SNA commands and data are placed in this buffer, and they can be in an Extended Binary Coded Decimal Interchange Code (EBCDIC) format.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are set for write functions and returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Supplied parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for RUI_WRITE and **SLI_SEND**, and returned by RUI_READ and RUI_BID. For the RH for **SLI_SEND**, all fields except the queued-response indicator (**lua_rh.qri**) and pacing indicator (**lua_rh.pi**) are used.

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits.

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set one of the following flags to 1 to indicate on which message flow the data is to be sent:

lua_flag1.sscp_exp

lua_flag1.sscp_norm

lua_flag1.lu_exp

lua_flag1.lu_norm

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. This is a supplied parameter for **SLI_SEND**.

Possible values are as follows:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SIGNAL

The SLI receives and responds to the BIND and STSN requests through the LUA interface extension routines.

LU-DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that RUI_BID was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_SEND** and should be set to zero.

lua_sequence_number

The union member of **LUA_SPECIFIC** used by **SLI_SEND**. Returned parameter. Contains the sequence number for either the first in the chain request unit or the only segment in the chain request unit. Note that this parameter is not byte-reversed.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_INVALID_FLOW

Secondary return code; the **lua_flag1** flow flags were set incorrectly when a verb was issued:

When issuing **SLI_SEND** to send an SNA response, set only one **lua_flag1** flow flag.

When issuing SLI_RECEIVE, set at least one lua_flag1 flow flag.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_REQUIRED_FIELD_MISSING

Secondary return code; the verb that was issued either did not include a data pointer (if the data count was not zero) or did not include an **lua_flag1** flow flag.

LUA_INVALID_MESSAGE_TYPE

Secondary return code; the **lua_message_type** parameter is not recognized by the LUA interface.

LUA_DATA_LENGTH_ERROR

Secondary return code; the application did not provide user-supplied data required by the verb issued. Note that when **SLI_SEND** is issued for an SNA LUSTAT command, status (in four bytes) is required, and that when SLI_OPEN is issued with secondary initialization, data is required.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_MAX_NUMBER_OF_SENDS

Secondary return code; the application issued a third **SLI_SEND** before one completed.

LUA_SEND_ON_FLOW_PENDING

Secondary return code; an **SLI_SEND** was still outstanding when the application issued another **SLI_SEND** for an SNA flow.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued, or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_DATA_SEGMENT_LENGTH_ERROR

Secondary return code; one of the following has occurred:

The supplied data segment for SLI_RECEIVE or **SLI_SEND** is not a read/write data segment as required.

The supplied data segment for SLI_RECEIVE is not as long as that provided in lua_max_length.

The supplied data segment for SLI_SEND is not as long as that provided in lua_data_length.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another OS/2 process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the LUA was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_INSUFFICIENT_RESOURCES

Secondary return code; a temporary condition of insufficient resources caused the request receiver to be unable to perform. The request sent to the half-session component was not executed, even though it was understood and supported.

LUA_SEND_CORRELATION_TABLE_FULL

Secondary return code; the session send correlation table for the flow requested reached its capacity.

LUA_RU_LENGTH_ERROR

Secondary return code; the request/response unit (RU) request was an incorrect length (either too short or too long). The request unit was not interpreted or processed even though it was delivered to the half-session component. The half-session capabilities do not match. This SNA sense code is also an exception request sense code.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_HDX_BRACKET_STATE_ERROR

Secondary return code; the existing state error prevented the current request from being sent. The determination was made by a protocol computer.

LUA_RESPONSE_ALREADY_SENT

Secondary return code; a response for the chain was already sent so that the current request was not sent. The determination was made by a protocol computer.

LUA_EXR_SENSE_INCORRECT

Secondary return code; the application responded negatively to an exception request. The sense code was unacceptable.

LUA_RESPONSE_OUT_OF_ORDER

Secondary return code; the current response was not for the oldest request. The determination was made by a protocol computer.

LUA_CHAIN_RESPONSE_REQUIRED

Secondary return code; a CHASE response was still outstanding when a more recent request was attempted. The determination was made by a protocol computer.

LUA_BRACKET

Secondary return code; the sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DIRECTION

Secondary return code; while the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_RESET

Secondary return code; a half-session of an active session but with inactive data traffic received a normal flow data flow control (DFC) or function management data (FMD) request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_QUIESCED

Secondary return code; a DFC or FMD request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_NOT_RESET

Secondary return code; while the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_NO_BEGIN_BRACKET

Secondary return code; the receiver has already sent a positive response to a BIS command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_SC_PROTOCOL_VIOLATION

Secondary return code; a violation of SC protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or data flow control state was found. Delivery to the half-session component was prevented.

LUA_IMMEDIATE_REQUEST_MODE_ERROR

Secondary return code; the request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_QUEUED_RESPONSE_ERROR

Secondary return code; the request violated the queued response protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_ERP_SYNC_EVENT_ERROR

Secondary return code; a violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_BEFORE_SENDING_REQ

Secondary return code; a previously received request has not been responded to yet and an attempt was made in half-duplex send/receive mode to send a normal flow request. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_CORRELATION_ERROR

Secondary return code; a response was sent that does not correspond to a previously received request or a response was received that does not correspond to a previously sent request.

LUA_BB_NOT_ALLOWED

Secondary return code; the begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EB_NOT ALLOWED

Secondary return code; the end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EXCEPTION_RSP_NOT_ALLOWED

Secondary return code; when an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_DEFINITE_RSP_NOT_ALLOWED

Secondary return code; when a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_ALLOWED

Secondary return code; the change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_RESPONSE_NOT_ALLOWED

Secondary return code; a request other than an EXR contained a "no response." The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CHAINING_NOT_SUPPORTED

Secondary return code; the chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_BRACKETS_NOT_SUPPORTED

Secondary return code; the bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_SUPPORTED

Secondary return code; the change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_FI

Secondary return code; the format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_ALTERNATE_CODE_NOT_SUPPORTED

Secondary return code; the code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_REQUEST_CODE

Secondary return code; the request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_SPEC_OF_SDI_RTI

Secondary return code; the SDI and the RTI were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_DR1I_DR2I_ERI

Secondary return code; the DR1I, the DR2I, and the ERI were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_QRI

Secondary return code; the queued response indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_EDI

Secondary return code; the EDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_PDI

Secondary return code; the PDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_SESSION

Secondary return code; a request to activate a session is required because no active half-session in the receiving end node for the origination-destination pair exists, or no active boundary function half-session component for the origination-destination pair in a node that supplies the boundary function exists. Delivery of the request could not take place for one of the following reasons:

A path information unit error

A path outage

An invalid sequence of requests for activation

If a path error is received during an active session, that usually indicates there is no longer a valid path to the session partner.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process has been canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation encountered an ABEND.

The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).

The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_SEND sends responses, SNA commands, and data from the Windows LUA application to a host LU. A session must already be open to issue **SLI_SEND** for a particular LU-LU session flow. To send data on the SSCP normal flow prior to the completion of SLI_OPEN, the session must have been initialized as primary with SSCP access. In addition, the session status must be INIT_COMPLETE.

The settings for lua_message_type determine the type of processing that will be done by SLI_SEND. The following table indicates the parameters to set based on the value of lua_message_type.

SLI_SEND parameter	LU_DATA SSCP_DATA	BID BIS RTR	CHASE QC	LUSTAT_LU LUSTAT_SSCP	QEC RELQ SBI SIGNAL	RQR	RSP
lua_data _length	Req.	0	0	Req.	0	0	Req. (0 if no data)
lua_data _ptr	Req. (0 if no data)	0	0	Req.	0	0	Req. (0 if no data)
lua_flag1 flow flags	0	0	0	0	0	0	Req. (set one)
lua_rh	FI DRL1 DRL2 RI BBI EBI CDI CSI EDI	SDI QRI	SDI QRI EBI CDI	SDI QRI DRL1 DRL2 RI BBI EBI CDI	SDI	0	RRI RI
lua_th	0	0	0	0	0	0	SNF

The location provided in **lua_data_ptr** and the length provided in **lua_data_length** determine the data that the SLI sends. The data will be chained by the SLI verbs if necessary.

When sending a response, the type of response determines the SLI_SEND information required. For all responses, you must:

- Set the selected **lua_flag1** flow flag.
- Provide the sequence number in lua_th.snf for the request to which you are responding.
- Set lua_message_type to LUA_MESSAGE_TYPE_RSP.

For multichain message responses, the sequence number of the last received chain element must be used. For a response to a multichain message ending with a CANCEL command, the CANCEL command sequence number is used.

For positive responses that only require the request code, set lua_rh.ri to zero (indicating that the response is positive) and lua_data_length to zero (indicating no data is provided). The request code is filled in by the SLI, using the sequence number provided.

For negative responses in which lua_rh.ri is set to 1, set the lua_data_ptr to the SNA sense code address and the lua_data_length to the SNA sense code length (four bytes). The sequence number is used by the SLI to fill in the request code.

See Also

Reference

RUI_INIT
RUI_READ
RUI_WRITE
SLI_BID
SLI_CLOSE
SLI_OPEN
SLI_RECEIVE

SLI_SEND_EX

The **SLI_SEND_EX** verb sends responses, SNA commands, and data from a Microsoft® Windows® logical unit application (LUA) application to a host logical unit (LU).

The SLI_SEND_EX verb also supports inbound chaining. The maximum length of data that can be sent by a single verb is 4,294,967,295 bytes. This is compared to a maximum of 65,535 bytes that can be sent by the SLI_SEND verb.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_SEND_EX.

The second syntax union below describes the **LUA_SPECIFIC** member of the VCB used by **SLI_SEND_EX**. Other union members are omitted for clarity.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
union LUA_SPECIFIC {
    struct SLI_SEND_EX_SPECIFIC {
        unsigned char lua_sequence_number[2];
        unsigned long lua_data_length_ex;
    };
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_SEND_EX.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_SEND only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_SEND_EX** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be set to zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_SEND_EX** and should be set to zero.

lua_data_length

This parameter is reserved and must be set to zero.

The length of data to be sent is set in the lua_data_length_ex parameter.

lua_data_ptr

Pointer to the application-supplied buffer that contains the data to be sent to the host by **SLI_SEND_EX**.

Both SNA commands and data are placed in this buffer, and they can be in an Extended Binary Coded Decimal Interchange Code (EBCDIC) format.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are set for write functions and returned for read and bid functions. Its subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Supplied parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for RUI_WRITE and **SLI_SEND**, and returned by RUI_READ and RUI_BID. For the RH for **SLI_SEND_EX**, all fields except the queued-response indicator (**lua_rh.qri**) and pacing indicator (**lua_rh.pi**) are used.

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

Request/response unit (RU) category, two bits.

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

System services control point (SSCP) expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

Set one of the following flags to 1 to indicate on which message flow the data is to be sent:

lua_flag1.sscp_exp

lua_flag1.sscp_norm

lua_flag1.lu_exp

lua_flag1.lu_norm

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. This is a supplied parameter for **SLI_SEND_EX**.

Possible values are as follows:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SIGNAL

The SLI receives and responds to the BIND and STSN requests through the LUA interface extension routines.

LU-DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that RUI_BID was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_SEND_EX** and should be set to zero.

lua_sequence_number

The union member of **LUA_SPECIFIC** used by **SLI_SEND_EX**. Returned parameter. Contains the sequence number for either the first in the chain request unit or the only segment in the chain request unit. Note that this parameter is not byte-reversed.

lua_data_length_ex

The union member of **LUA_SPECIFIC** used by **SLI_SEND_EX**. Supplied parameter. Specifies the length of data being sent.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_PARAMETER_CHECK

Primary return code; the verb did not execute because of a parameter error.

LUA_INVALID_LUNAME

Secondary return code; an invalid **lua_luname** was specified.

LUA_BAD_SESSION_ID

Secondary return code; an invalid value for **lua_sid** was specified in the VCB.

LUA_BAD_DATA_PTR

Secondary return code; the **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment and supplied data is required.

LUA_RESERVED_FIELD_NOT_ZERO

Secondary return code; a reserved parameter for the verb just issued is not set to zero.

LUA_INVALID_POST_HANDLE

Secondary return code; for a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows LUA VCB does not contain a valid event handle.

LUA_INVALID_FLOW

Secondary return code; the **lua_flag1** flow flags were set incorrectly when a verb was issued:

When issuing **SLI_SEND_EX** to send an SNA response, set only one **lua_flag1** flow flag.

When issuing SLI_RECEIVE_EX, set at least one lua_flag1 flow flag.

LUA_VERB_LENGTH_INVALID

Secondary return code; an LUA verb was issued with a value for **lua_verb_length** unexpected by LUA.

LUA_REQUIRED_FIELD_MISSING

Secondary return code; the verb that was issued either did not include a data pointer (if the data count was not zero) or did not include an **lua_flag1** flow flag.

LUA_INVALID_MESSAGE_TYPE

Secondary return code; the **lua_message_type** parameter is not recognized by the LUA interface.

LUA_DATA_LENGTH_ERROR

Secondary return code; the application did not provide user-supplied data required by the verb issued. Note that when **SLI_SEND_EX** is issued for an SNA LUSTAT command, status (in four bytes) is required, and that when SLI_OPEN is issued with secondary initialization, data is required.

LUA_STATE_CHECK

Primary return code; the verb did not execute because it was issued in an invalid state.

LUA_NO_SLI_SESSION

Secondary return code; a session was not open or was down due to an SLI_CLOSE or session failure when a command was issued.

LUA_MAX_NUMBER_OF_SENDS

Secondary return code; the application issued a third **SLI_SEND** or an **SLI_SEND_EX** before one completed.

LUA_SEND_ON_FLOW_PENDING

Secondary return code; an **SLI_SEND** or an **SLI_SEND_EX** was still outstanding when the application issued another **SLI_SEND_EX** for an SNA flow.

LUA_SESSION_FAILURE

Primary return code; an error condition, specified in the secondary return code, caused the session to fail.

LUA_RECEIVED_UNBIND

Secondary return code; the primary logical unit (PLU) sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_NO_RUI_SESSION

Secondary return code; no session has been initialized for the LUA verb issued, or some verb other than SLI_OPEN was issued before the session was initialized.

LUA_LU_COMPONENT_DISCONNECTED

Secondary return code; an LU component is unavailable because it is not connected properly. Make sure that the power is on.

LUA_DATA_SEGMENT_LENGTH_ERROR

Secondary return code; one of the following has occurred:

The supplied data segment for SLI_RECEIVE_EX or **SLI_SEND_EX** is not a read/write data segment as required.

The supplied data segment for SLI_RECEIVE_EX is not as long as that provided in lua_max_length_ex.

The supplied data segment for SLI_SEND_EX is not as long as that provided in lua_data_length_ex.

LUA_VERB_RECORD_SPANS_SEGMENTS

Secondary return code; the LUA VCB length parameter plus the segment offset is beyond the segment end.

LUA_NOT_ACTIVE

Secondary return code; LUA was not active within Microsoft Host Integration Server or SNA Server when an LUA verb was issued.

LUA_SLI_LOGIC_ERROR

Secondary return code; the LUA interface found an internal error in logic.

LUA_INVALID_PROCESS

Secondary return code; the session for which an LUA verb was issued is unavailable because another OS/2 process owns the session.

LUA_LU_INOPERATIVE

Secondary return code; a severe error occurred while the LUA was attempting to stop the session. This LU is unavailable for any LUA requests until an activate logical unit (ACTLU) is received from the host.

LUA_MODE_INCONSISTENCY

Secondary return code; performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

LUA_INSUFFICIENT_RESOURCES

Secondary return code; a temporary condition of insufficient resources caused the request receiver to be unable to perform. The request sent to the half-session component was not executed, even though it was understood and supported.

LUA_SEND_CORRELATION_TABLE_FULL

Secondary return code; the session send correlation table for the flow requested reached its capacity.

LUA_RU_LENGTH_ERROR

Secondary return code; the RU request was an incorrect length (either too short or too long). The request unit was not interpreted or processed even though it was delivered to the half-session component. The half-session capabilities do not match. This SNA sense code is also an exception request sense code.

LUA_FUNCTION_NOT_SUPPORTED

Secondary return code; LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

LUA_HDX_BRACKET_STATE_ERROR

Secondary return code; the existing state error prevented the current request from being sent. The determination was made by a protocol computer.

LUA_RESPONSE_ALREADY_SENT

Secondary return code; a response for the chain was already sent so that the current request was not sent. The determination was made by a protocol computer.

LUA_EXR_SENSE_INCORRECT

Secondary return code; the application responded negatively to an exception request. The sense code was unacceptable.

LUA_RESPONSE_OUT_OF_ORDER

Secondary return code; the current response was not for the oldest request. The determination was made by a protocol

computer.

LUA_CHAIN_RESPONSE_REQUIRED

Secondary return code; a CHASE response was still outstanding when a more recent request was attempted. The determination was made by a protocol computer.

LUA_BRACKET

Secondary return code; the sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DIRECTION

Secondary return code; while the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_RESET

Secondary return code; a half-session of an active session but with inactive data traffic received a normal flow data flow control (DFC) or function management data (FMD) request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_QUIESCED

Secondary return code; a DFC or FMD request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_DATA_TRAFFIC_NOT_RESET

Secondary return code; while the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_NO_BEGIN_BRACKET

Secondary return code; the receiver has already sent a positive response to a BIS command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_SC_PROTOCOL_VIOLATION

Secondary return code; a violation of SC protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or data flow control state was found. Delivery to the half-session component was prevented.

LUA_IMMEDIATE_REQUEST_MODE_ERROR

Secondary return code; the request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_QUEUED_RESPONSE_ERROR

Secondary return code; the request violated the queued response protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_ERP_SYNC_EVENT_ERROR

Secondary return code; a violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_BEFORE_SENDING_REQ

Secondary return code; a previously received request has not been responded to yet and an attempt was made in half-duplex send/receive mode to send a normal flow request. An invalid header request or request unit for the received current session

control or data flow control state was found. Delivery to the half-session component was prevented.

LUA_RSP_CORRELATION_ERROR

Secondary return code; a response was sent that does not correspond to a previously received request or a response was received that does not correspond to a previously sent request.

LUA_BB_NOT_ALLOWED

Secondary return code; the begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EB_NOT ALLOWED

Secondary return code; the end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_EXCEPTION_RSP_NOT_ALLOWED

Secondary return code; when an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_DEFINITE_RSP_NOT_ALLOWED

Secondary return code; when a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_ALLOWED

Secondary return code; the change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_RESPONSE_NOT_ALLOWED

Secondary return code; a request other than an EXR contained a "no response." The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CHAINING_NOT_SUPPORTED

Secondary return code; the chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_BRACKETS_NOT_SUPPORTED

Secondary return code; the bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_CD_NOT_SUPPORTED

Secondary return code; the change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_FI

Secondary return code; the format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_ALTERNATE_CODE_NOT_SUPPORTED

Secondary return code; the code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_REQUEST_CODE

Secondary return code; the request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_SPEC_OF_SDI_RTI

Secondary return code; the SDI and the RTI were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_DR1I_DR2I_ERI

Secondary return code; the DR1I, the DR2I, and the ERI were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_QRI

Secondary return code; the queued response indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_EDI

Secondary return code; the EDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_INCORRECT_USE_OF_PDI

Secondary return code; the PDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

LUA_NO_SESSION

Secondary return code; a request to activate a session is required because no active half-session in the receiving end node for the origination-destination pair exists, or no active boundary function half-session component for the origination-destination pair in a node that supplies the boundary function exists. Delivery of the request could not take place for one of the following reasons:

A path information unit error

A path outage

An invalid sequence of requests for activation

If a path error is received during an active session, that usually indicates there is no longer a valid path to the session partner.

LUA_CANCELED

Primary return code; the secondary return code gives the reason for canceling the command.

LUA_TERMINATED

Secondary return code; the session was terminated when a verb was pending. The verb process has been canceled.

LUA_IN_PROGRESS

Primary return code; an asynchronous command was received but is not completed.

LUA_COMM_SUBSYSTEM_ABENDED

Primary return code; indicates one of the following conditions:

The node used by this conversation encountered an ABEND.

The connection between the transaction program (TP) and the physical unit (PU) 2.1 node has been broken (a LAN error).

The SnaBase at the TPs computer encountered an ABEND.

LUA_COMM_SUBSYSTEM_NOT_LOADED

Primary return code; a required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

LUA_INVALID_VERB_SEGMENT

Primary return code; the VCB extended beyond the end of the data segment.

LUA_UNEXPECTED_DOS_ERROR

Primary return code; after issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

LUA_STACK_TOO_SMALL

Primary return code; the stack size of the application is too small to execute the verb. Increase the stack size of your application.

LUA_INVALID_VERB

Primary return code; either the verb code or the operation code, or both, is invalid. The verb did not execute.

Remarks

SLI_SEND_EX sends responses, SNA commands, and data from the Windows LUA application to a host LU.

The difference between SLI_SEND_EX and SLI_SEND is that the SLI_SEND_EX verb supports inbound chaining and can send up to a maximum of 4,295 kilobytes (KB) in a single verb request. In contrast, SLI_SEND is limited to sending up to 64 KB in a verb request. A single SLI_SEND_EX or SLI_SEND verb defines a chain. A single SLI_RECEIVE_EX or SLI_RECEIVE verb receives a whole chain.

A session must already be open to issue SLI_SEND_EX for a particular LU-LU session flow. To send data on the SSCP normal flow prior to the completion of SLI_OPEN, the session must have been initialized as primary with SSCP access. In addition, the session status must be INIT_COMPLETE.

The settings for lua_message_type determine the type of processing that will be done by SLI_SEND_EX. The following table indicates the parameters to set based on the value of lua_message_type.

SLI_SEND_EX parameter	**LU_DATA SSCP_DATA**	**BID BIS RTR**	**CHASE QC**	**LUSTAT_LU LUSTAT_SSCP**	**QEC RELQ SBI SIGNAL**	**RQR**	**RSP**
lua_data _length	Req.	0	0	Req.	0	0	Req. (0 if no data)
lua_data _ptr	Req. (0 if no data)	0	0	Req.	0	0	Req. (0 if no data)
lua_flag1 flow flags	0	0	0	0	0	0	Req. (set one)
lua_rh	FI DRL1 DRL2 RI BBI EBI CDI CSI EDI	SDI QRI	SDI QRI EBI CDI	SDI QRI DRL1 DRL2 RI BBI EBI CDI	SDI	0	RRI RI

lua_th	0	0	0	0	0	0	SNF

The location provided in **lua_data_ptr** and the length provided in **lua_data_length_ex** determine the data that the SLI sends. The data will be chained by the SLI verbs if necessary.

When sending a response, the type of response determines the SLI_SEND_EX information required. For all responses, you must:

- Set the selected **lua_flag1** flow flag.
- Provide the sequence number in lua_th.snf for the request to which you are responding.
- Set lua_message_type to LUA_MESSAGE_TYPE_RSP.

For multichain message responses, the sequence number of the last received chain element must be used. For a response to a multichain message ending with a CANCEL command, the CANCEL command sequence number is used.

For positive responses that only require the request code, set lua_rh.ri to zero (indicating that the response is positive) and lua_data_length to zero (indicating no data is provided). The request code is filled in by the SLI, using the sequence number provided.

For negative responses in which lua_rh.ri is set to 1, set the lua_data_ptr to the SNA sense code address and the lua_data_length to the SNA sense code length (four bytes). The sequence number is used by the SLI to fill in the request code.

See Also

Reference

RUI_INIT
RUI_READ
RUI_WRITE
SLI_BID
SLI_CLOSE
SLI_OPEN
SLI_RECEIVE_EX

SLI_BIND_ROUTINE

The **SLI_BIND_ROUTINE** verb notifies the Microsoft® Windows® logical unit application (LUA) application that a BIND request has come from the host and allows the user-supplied routine to examine the request and formulate a response.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_BIND_ROUTINE.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_BIND_ROUTINE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_BIND_ROUTINE only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_BIND_ROUTINE** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_BIND_ROUTINE** and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of the BIND request/response unit (RU) data returned in the data buffer.

lua_data_ptr

For the **SLI_BIND_ROUTINE** this parameter contains the address of the BIND RU.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Supplied parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are returned for read and bid functions.

lua_rh

Supplied parameter. Contains the SNA request/response header (RH) of the message sent or received.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application.

lua_message_type

Supplied parameter. Specifies the type of SNA data or command sent to the host.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_BIND_ROUTINE** and should be set to zero.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_NEGATIVE_RSP

Primary return code; either the LUA sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

Remarks

SLI_BIND_ROUTINE provides a mechanism for the Windows LUA application to examine BIND requests that are received from the host. The Windows LUA uses a user-supplied dynamic-link library (DLL) to notify the Windows LUA application that a BIND request has been received. The user-supplied DLL routine then examines the contents of the BIND and formulates a response for the request.

The DLL name for the routine is provided as extensions of the SLI_OPEN verbs VCB. The lua_extension_list_offset parameter provides the offset from the start of the VCB to the first name in the extension list.

The Windows LUA interface assigns storage space where the VCB is structured. The VCB of SLI_BIND_ROUTINE contains lua_th and lua_rh. The address of the BIND RU is specified in lua_data_ptr and the length of the RU is specified in lua_data_length.

When SLI_BIND_ROUTINE returns to the Windows LUA, processing of SLI_BIND_ROUTINE is completed. The BIND response should overwrite the BIND RU. When the BIND is accepted, the primary return code should be set to LUA_OK. If the BIND is rejected, the primary return code should be set to LUA_NEGATIVE_RSP and the BIND buffer contains the negative sense code. The lua_data_ptr parameter should not be modified.

If a negative response is returned from SLI_BIND_ROUTINE, SLI_OPEN is canceled. The lua_prim_rc of the SLI_OPEN is set to LUA_SESSION_FAILURE, and the lua_sec_rc is set to LUA_NEG_RSP_FROM_BIND_ROUTINE.

See Also

Reference

RUI_INIT
RUI_PURGE
RUI_READ
RUI_WRITE
SLI_OPEN
SLI_PURGE
SLI_RECEIVE
SLI_SEND

SLI_STSN_ROUTINE

The **SLI_STSN_ROUTINE** verb notifies the Microsoft® Windows® logical unit application (LUA) application that an STSN command has come from the host and allows the user-supplied routine to examine the request and formulate a response.

The following structure describes the LUA_COMMON member of the verb control block (VCB) used by SLI_STSN_ROUTINE.

Syntax

```
struct LUA_COMMON {
    unsigned short    lua_verb;
    unsigned short    lua_verb_length;
    unsigned short    lua_prim_rc;
    unsigned long     lua_sec_rc;
    unsigned short    lua_opcode;
    unsigned long     lua_correlator;
    unsigned char     lua_luname[8];
    unsigned short    lua_extension_list_offset;
    unsigned short    lua_cobol_offset;
    unsigned long     lua_sid;
    unsigned short    lua_max_length;
    unsigned short    lua_data_length;
    char FAR *        lua_data_ptr;
    unsigned long     lua_post_handle;
    struct LUA_TH     lua_th;
    struct LUA_RH     lua_rh;
    struct LUA_FLAG1  lua_flag1;
    unsigned char     lua_message_type;
    struct LUA_FLAG2  lua_flag2;
    unsigned char     lua_resv56[7];
    unsigned char     lua_encr_decr_option;
};
```

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_SLI for Session Level Interface (SLI) verbs.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, LUA_OPCODE_SLI_STSN_ROUTINE.

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_STSN_ROUTINE only requires this parameter if lua_sid is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Not used by **SLI_STSN_ROUTINE** and should be set to zero.

lua_cobol_offset

Not used by LUA in Microsoft® Host Integration Server or SNA Server and should be zero.

lua_sid

Supplied parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Not used by **SLI_STSN_ROUTINE** and should be set to zero.

lua_data_length

Returned parameter. Specifies the length of the STSN request/response unit (RU) data returned in the data buffer.

lua_data_ptr

For the **SLI_STSN_ROUTINE** this parameter contains the address of the STSN RU.

lua_post_handle

Supplied parameter. Used under Microsoft Windows Server™ 2003 or Windows 2000 Server if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

For all other environments, this parameter is reserved and should be set to zero.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message received. Various subparameters are returned for read and bid functions.

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application.

lua_message_type

Supplied parameter. Specifies the type of SNA data or command sent to the host.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates system services control point (SSCP) expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

Reserved and should be set to zero.

lua_encr_decr_option

Not used by **SLI_STSN_ROUTINE** and should be set to zero.

Return Codes

LUA_OK

Primary return code; the verb executed successfully.

LUA_SEC_OK

Secondary return code; no additional information exists for LUA_OK.

LUA_NEGATIVE_RSP

Primary return code; either the LUA sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

Remarks

SLI_STSN_ROUTINE provides a mechanism for the Windows LUA application to examine and respond to STSN commands. The Windows LUA notifies the Windows LUA application that an STSN command has been received from the host. This is done through a user-supplied dynamic-link library (DLL). The users DLL examines the STSN request and formulates a response to the request.

The DLL name for the routine is provided as extensions of the SLI_OPEN verbs VCB. The lua_extension_list_offset parameter provides the offset from the start of the VCB to the first name in the extension list.

The Windows LUA interface assigns storage space where the VCB is structured. The VCB of the SLI_STSN_ROUTINE contains lua_th and lua_rh. The address of the STSN RU is specified in lua_data_ptr and the length of the RU is specified in lua_data_length.

When SLI_STSN_ROUTINE returns to the Windows LUA, processing of the SLI_STSN_ROUTINE is completed. The STSN response should overwrite the STSN RU. When the STSN is accepted, the primary return code should be set to LUA_OK. If the STSN is rejected, the primary return code should be set to LUA_NEGATIVE_RSP and the STSN buffer contains the negative sense code. The lua_data_ptr parameter should not be modified.

If a negative response is returned from SLI_STSN_ROUTINE, SLI_OPEN is canceled. The lua_prim_rc of the SLI_OPEN is set to LUA_SESSION_FAILURE, and the lua_sec_rc is set to LUA_NEG_RSP_FROM_STSN_ROUTINE.

See Also

Reference

RUI_INIT
RUI_PURGE
RUI_READ
RUI_WRITE
SLI_OPEN
SLI_PURGE
SLI_RECEIVE
SLI_SEND

LUA Extensions for the Windows Environment

The extensions described in this section are designed for Microsoft® Windows®. They provide support for programming compatibility and optimum application performance in 32-bit operating environments. These extensions are supported on Microsoft Windows Server™ 2003, Windows XP Professional, and Windows 2000 Server.

The Windows logical unit application (LUA) programming interface enables multithreaded Windows-based processes. A process contains one or more threads of execution. For each extension, this section provides a definition of the function with syntax, return codes, and remarks for using the extension.

These functions can be grouped into two categories depending on whether Request Unit Interface (RUI) or Session Level Interface (SLI) verbs are used.

In This Section

RUI

The **RUI** function provides event notification for all Request Unit Interface (RUI) verbs.

Syntax

```
            void WINAPI RUI(
    LUA_VERB_RECORD FAR *lpVCB  );
```

Parameters

lpVCB

Pointer to the logical unit application (LUA) verb control block (VCB), **LUA_VERB_RECORD**.

Return Value

The code returned in **lua_prim_rc** indicates whether asynchronous notification will occur. If the field is set to LUA_IN_PROGRESS, asynchronous notification will occur through event signaling. If the flag is not LUA_IN_PROGRESS, the request completed synchronously. Examine the primary return code and secondary return code for any errors.

Remarks

The application must provide a handle to an event in the **lua_post_handle** parameter of the VCB. The event must be in the not-signaled state.

When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

See Also

Reference

WinRUI

SLI

The **SLI** function provides event notification for all Session Level Interface (SLI) verbs.

Syntax

```
            void WINAPI SLI(
    LUA_VERB_RECORD FAR *lpVCB  );
```

Parameters

lpVCB

Pointer to the logical unit application (LUA) verb control block (VCB), **LUA_VERB_RECORD**.

Return Value

The code returned in **lua_prim_rc** indicates whether asynchronous notification will occur. If the field is set to LUA_IN_PROGRESS, asynchronous notification will occur through event signaling. If the flag is not LUA_IN_PROGRESS, the request completed synchronously. Examine the primary return code and secondary return code for any errors.

Remarks

The application must provide a handle to an event in the **lua_post_handle** parameter of the VCB. The event must be in the not-signaled state.

When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

See Also

Reference

WinSLI

WinRUI

The **WinRUI** function provides asynchronous message notification for all Microsoft® Windows®-based Request Unit Interface (RUI) verbs.

Syntax

```
int WINAPI WinRUI(
HWND hWnd,
LUA_VERB_RECORD FAR *lpVCB
);
```

Parameters

hWnd

Handle of window to receive message.

lpVCB

Pointer to the logical unit application (LUA) verb control block (VCB), **LUA_VERB_RECORD**.

Return Value

The function returns a value indicating whether the request was accepted by the Windows-based RUI for processing. A returned value of zero indicates that the request was accepted and will be processed. A value other than zero indicates an error. Possible error codes are as follows:

WLUAINVALIDHANDLE

The window handle provided is invalid.

WLUASTARTUPNOTCALLED

The application has not initiated a session using WinRUIStartup.

The value returned in **lua_flag2.async** indicates whether asynchronous notification will occur. If the flag is set (nonzero), asynchronous notification will occur through a message posted to the applications message queue. If the flag is not set, the request completed synchronously. Examine the primary return code and secondary return code for any error conditions.

Remarks

When the asynchronous operation is complete, the applications window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinRUI" as the input string. The *lParam* argument contains the address of the VCB being posted as complete. The *wParam* argument is undefined.

Note

It is possible for the request to be accepted for processing (the function call returns zero) but rejected later with a primary return code and secondary return code set in the VCB. Examine the primary return code and secondary return code for any error conditions.

If the application calls **WinRUI** without first initializing the session using **WinRUIStartup**, an error is returned.

See Also

Reference

RUI

WinRUIStartup

WinRUICleanup

The **WinRUICleanup** function terminates and deregisters an application using Request Unit Interface (RUI) verbs from a Microsoft® Windows® logical unit application (LUA) implementation.

Syntax

```
BOOL WINAPI WinRUICleanup(void);
```

Return Value

The return code specifies whether the deregistration was successful. If the value is not zero, the application was successfully deregistered. If the value is zero, the application was not deregistered.

Remarks

Use **WinRUICleanup** to indicate deregistration of a Windows LUA application from a Windows LUA implementation. This function can be used, for example, to free up resources allocated to the specific application.

If **WinRUICleanup** is called while LUs are in session (RUI_TERM not issued), the cleanup code should issue an **RUI_TERM** close type ABEND for the application for all open sessions.

See Also

Reference

RUI_TERM

WinRUIStartup

WinRUIGetLastInitStatus

The **WinRUIGetLastInitStatus** function enables an application to determine the status of an RUI_INIT, so that the application can evaluate whether the **RUI_INIT** should be timed out. This extension can be used to initiate status reporting, terminate status reporting, or find the current status. For details, see the Remarks section.

Syntax

```
          int WINAPI WinRUIGetLastInitStatus(
    DWORD dwSid,
     HANDLE hStatusHandle,
    DWORD dwNotifyType,
    BOOL bClearPrevious    );
```

Parameters

dwSid

Specifies the RUI session identifier of the session for which status will be determined. If *dwSid* is zero, *hStatusHandle* is used to report status on all sessions. Note that the **lua_sid** parameter in the RUI_INIT verb control block (VCB) is valid as soon as the call to RUI or WinRUI for the **RUI_INIT** returns.

hStatusHandle

Specifies a handle used for signaling the application that the status for the session (specified by *dwSid*) has changed. Can be a window handle, an event handle, or NULL; *dwNotifyType* must be set accordingly:

If *hStatusHandle* is a window handle, status is sent to the application through a window message. The message is obtained from **RegisterWindowMessage** using the string "WinRUI". The parameter *wParam* contains the session status. (For more information, see Return Codes.) Depending on the value of *dwNotifyType*, *lParam* contains either the RUI session identifier of the session, or the value of **lua_correlator** from the RUI_INIT verb.

If *hStatusHandle* is an event handle, when the status for the session specified by *dwSid* changes, the event is put into the signaled state. The application must then make a further call to **WinRUIGetLastInitStatus** to find out the new status. Note that the event should not be the same as one used for signaling completion of any RUI verb.

If *hStatusHandle* is NULL, the status of the session specified by *dwSid* is returned in the return code. In this case, *dwSid* must not be zero unless *bClearPrevious* is TRUE. If *hStatusHandle* is NULL, *dwNotifyType* is ignored.

dwNotifyType

Specifies the type of indication required. This determines the contents of the *lParam* of the window message, and how **WinRUIGetLastInitStatus** interprets *hStatusHandle*. Allowed values are:

WLUA_NTFY_EVENT

The *hStatusHandle* parameter contains an event handle.

WLUA_NTFY_MSG_CORRELATOR

The *hStatusHandle* parameter contains a window handle, and the *lParam* of the returned window message should contain the value of the **lua_correlator** field on the RUI_INIT.

WLUA_NTFY_MSG_SID

The *hStatusHandle* parameter contains a window handle, and the *lParam* of the returned window message should contain the LUA session identifier.

bClearPrevious

If TRUE, status messages are no longer sent for the session identified by *dwSid*. If *dwSid* is zero, status messages are no longer sent for any session. If *bClearPrevious* is TRUE, *hStatusHandle* and *dwNotifyType* are ignored.

Return Value

WLUASYSNOTREADY

SNABASE is not running.

WLUANTFYINVALID

The *dwNotifyType* parameter is invalid.

WLUAINVALIDHANDLE

The *hStatusHandle* parameter does not contain a valid handle.

WLUASTARTUPNOTCALLED

WinRUIStartup has not been called.

WLUALINKINACTIVE

The link to the host is not yet active.

WLUALINKACTIVATING

The link to the host is being activated.

WLUAPUINACTIVE

The link to the host is active, but no ACTPU has yet been received.

WLUAPUACTIVE

An ACTPU has been received.

WLUAPUREACTIVATED

The physical unit (PU) has been reactivated.

WLUALUINACTIVE

The link to the host is active, and an ACTPU has been received, but no ACTLU has been received.

WLUALUACTIVE

The LU is active.

WLUALUREACTIVATED

The LU has been reactivated.

WLUAUNKNOWN

The session is in an unknown status. (This is an internal error.)

WLUAGETLU

The session is waiting for an Open(SSCP) response from the node.

WLUASIDINVALID

The security ID (SID) specified does not match any known by the RUI.

WLUASIDZERO

The *hStatusHandle* parameter is NULL and *bClearPrevious* is FALSE, but *dwSid* is zero.

WLUAGLOBALHANDLER

The *dwSid* parameter is zero, and messages from all sessions will be notified. (This is a normal return code, not an error.)

Remarks

This extension is intended to be used with either a window handle or an event handle to enable asynchronous notification of status changes. It can also be used alone to find the current status of a session.

With a window handle

There are two ways to use this extension with a window handle:

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_CORRELATOR,FALSE);
```

—or—

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_SID,FALSE);
```

With this implementation, changes in status are reported by a window message sent to the window handle specified. If WLUA_NTFY_MSG_CORRELATOR is specified, the *lParam* field in the window message contains the **lua_correlator** field for the session. If WLUA_NTFY_MSG_SID is specified, the *lParam* field in the window message contains the LUA session identifier for the session.

When the extension has been used with a window handle, use the following to cancel status reporting:

```
WinRUIGetLastInitStatus(Sid,NULL,0,TRUE);
```

For this implementation, note that if *Sid* is nonzero, status is only reported for that session. If *Sid* is zero, status is reported for all sessions.

With an event handle

To use this extension with an event handle, implement it as follows:

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NOTIFY_EVENT,FALSE);
```

The event whose handle is given will be signaled when a change in state occurs. Because no information is returned when an event is signaled, a further call must be issued to find out the status.

```
Status = WinRUIGetLastInitStatus(Sid,NULL,0,0,FALSE);
```

Note that in this case, a *Sid* must be specified.

When the extension has been used with an event handle, use the following to cancel the reporting of status:

```
WinRUIGetLastInitStatus(Sid,NULL,0,TRUE);
```

Query current status

To use this extension to query the current status of a session, it is not necessary to use an event or window handle. Instead, use the following:

```
Status = WinRUIGetLastInitStatus(Sid,NULL,0,0,FALSE);
```

See Also

Reference

RUI
RUI_INIT
WinRUI
WinRUIStartup

WinRUIStartup

The **WinRUIStartup** function enables an application using Request Unit Interface (RUI) verbs to specify the version of Windows logical unit application (LUA) required and to retrieve details of the specific Microsoft Windows LUA implementation. This function must be called by an application to register itself with a Windows LUA implementation before issuing any further Windows LUA calls.

Syntax

```
            int WINAPI WinRUIStartup(
    WORD wVersionRequired,
      LUADATA FAR *lpLuaData  );
```

Parameters

wVersionRequired

Specifies the version of Windows LUA support required. The high-order byte specifies the minor version (revision) number. The low-order byte specifies the major version number.

lpLuaData

Pointer to the **LUADATA** structure containing the returned version number information.

Return Value

The return code specifies whether the application was registered successfully and whether the Windows LUA implementation can support the specified version number. If the value is zero, it was registered successfully and the specified version can be supported. Otherwise, the return code is one of the following:

WLUASYSNOTREADY

The underlying network system is not ready for network communication.

WLUAVERNOTSUPPORTED

The version of Windows LUA support requested is not provided by this particular Windows LUA implementation.

WLUAINVALID

The Windows LUA version specified by the application is not supported by this dynamic-link library (DLL).

WLUAFAILURE

A failure occurred while the Windows LUA DLL was initializing. This usually occurs because an operating system call failed.

WLUAINITREJECT

WinRUIStartup was called multiple times.

Remarks

To support future Windows LUA implementations and applications that may have functionality differences, a negotiation takes place in **WinRUIStartup**. An application passes to **WinRUIStartup** the Windows LUA version that it can use. If this version is lower than the lowest version supported by the Windows LUA DLL, the DLL cannot support the application and **WinRUIStartup** fails. If the version is not lower, however, the call succeeds and returns the highest version of Windows LUA supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows LUA DLL on the system.

This negotiation allows both a Windows LUA DLL and a Windows LUA application to support a range of Windows LUA versions. An application can successfully use a DLL if there is any overlap in the versions. The following table illustrates how **WinRUIStartup** works in conjunction with different application and DLL versions.

LUA App versions	LUA DLL versions	To WinRUIStartup	From WinRUIStartup	Result
1.0	1.0	1.0	1.0	Use 1.0

1.0, 2.0	1.0	2.0	1.0	Use 1.0
1.0	1.0, 2.0	1.0	2.0	Use 1.0
1.0	2.0, 3.0	1.0	WLUAINVALID	Fail
2.0, 3.0	1.0	3.0	1.0	App fails
1.0, 2.0, 3.0	1.0, 2.0, 3.0	3.0	3.0	Use 3.0

Note

The application that uses RUI verbs must call **WinRUIStartup** prior to issuing any other LUA commands. However, **WinRUIStartup** needs to be called only once per application. If it is called multiple times, the subsequent calls will be rejected.

Details of the actual LUA implementation are described in the **WLUADATA** structure, defined as follows:

```
typedef struct { WORD wVersion;
            char szDescription[WLUADESCRIPTION_LEN+1];
            } LUADATA;
```

Having made its last Windows LUA call, an application should call the **WinRUICleanup** routine.

Each LUA application that uses RUI verbs must make a **WinRUIStartup** call before issuing any other LUA calls.

See Also

Reference

WinRUICleanup

WinSLI

The **WinSLI** function provides asynchronous message notification for all Microsoft® Windows®-based Session Level Interface (SLI) verbs.

Syntax

```
            int WINAPI WinSLI(
    HWND hWnd,
      LUA_VERB_RECORD FAR *lpVCB  );
```

Parameters

hWnd

Handle of window to receive message.

lpVCB

Pointer to the logical unit application (LUA) verb control block (VCB), **LUA_VERB_RECORD**.

Return Value

The function returns a value indicating whether the request was accepted by the Windows-based SLI for processing. A returned value of zero indicates that the request was accepted and will be processed. A value other than zero indicates an error. Possible error codes are as follows:

WLUAINVALIDHANDLE

The window handle provided is invalid.

WLUASTARTUPNOTCALLED

The application has not initiated a session using WinSLIStartup.

The value returned in **lua_flag2.async** indicates whether asynchronous notification will occur. If the flag is set (nonzero), asynchronous notification will occur through a message posted to the applications message queue. If the flag is not set, the request completed synchronously. Examine the primary return code and secondary return code for any error conditions.

Remarks

When the asynchronous operation is complete, the applications window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinSLI" as the input string. The *lParam* argument contains the address of the VCB being posted as complete. The *wParam* argument is undefined.

Note

It is possible for the request to be accepted for processing (the function call returns zero) but rejected later with a primary return code and secondary return code set in the VCB. Examine the primary return code and secondary return code for any error conditions.

If the application calls **WinSLI** without first initializing the session using **WinSLIStartup**, an error is returned.

See Also

Reference

SLI

WinSLIStartup

WinSLICleanup

The **WinSLICleanup** function terminates and deregisters an application using Session Level Interface (SLI) verbs from a Microsoft® Windows® logical unit application (LUA) implementation.

Syntax

```
BOOL WINAPI WinSLICleanup(void);
```

Return Value

The return code specifies whether the deregistration was successful. If the value is not zero, the application was successfully deregistered. If the value is zero, the application was not deregistered.

Remarks

Use **WinSLICleanup** to indicate deregistration of a Windows LUA application from a Windows LUA implementation. This function can be used, for example, to free up resources allocated to the specific application.

If **WinSLICleanup** is called while LUs are in session (SLI_CLOSE not issued), the cleanup code should issue an **SLI_CLOSE** close type ABEND for the application for all open sessions.

See Also

Reference

SLI_CLOSE

WinSLIStartup

WinSLIStartup

The **WinSLIStartup** function allows an application using the Session Level Interface (SLI) verbs to specify the version of Microsoft Windows logical unit application (LUA) required and to retrieve details of the specific Windows LUA implementation. This function must be called by an application to register itself with a Windows LUA implementation before issuing any further Windows LUA calls.

Syntax

```
int WINAPI WinSLIStartup(
WORD wVersionRequired,
LUADATA FAR *lpLuaData
);
```

Parameters

wVersionRequired

Specifies the version of Windows LUA support required. The high-order byte specifies the minor version (revision) number. The low-order byte specifies the major version number.

lpLuaData

Pointer to the **LUADATA** structure containing the returned version number information.

Return Value

The return code specifies whether the application was registered successfully and whether the Windows LUA implementation can support the specified version number. If the value is zero, it was registered successfully and the specified version can be supported. Otherwise, the return code is one of the following:

WLUASYSNOTREADY

The underlying network system is not ready for network communication.

WLUAVERNOTSUPPORTED

The version of Windows LUA support requested is not provided by this particular Windows LUA implementation.

WLUAINVALID

The Windows LUA version specified by the application is not supported by this dynamic-link library (DLL).

WLUAFAILURE

A failure occurred while the Windows LUA DLL was initializing. This usually occurs because an operating system call failed.

WLUAINITREJECT

WinSLIStartup was called multiple times.

Remarks

To support future Windows LUA implementations and applications that may have functionality differences, a negotiation takes place in **WinSLIStartup**. An application passes to **WinSLIStartup** the Windows LUA version that it can use. If this version is lower than the lowest version supported by the Windows LUA DLL, the DLL cannot support the application and **WinSLIStartup** fails. If the version is not lower, however, the call succeeds and returns the highest version of Windows LUA supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows LUA DLL on the system.

This negotiation allows both a Windows LUA DLL and a Windows LUA application to support a range of Windows LUA versions. An application can successfully use a DLL if there is any overlap in the versions. The following table illustrates how **WinSLIStartup** works in conjunction with different application and DLL versions.

App versions	LUA DLL versions	To WinSLIStartup	From WinSLIStartup	Result
1.0	1.0	1.0	1.0	Use 1.0

1.0, 2.0	1.0	2.0	1.0	Use 1.0
1.0	1.0, 2.0	1.0	2.0	Use 1.0
1.0	2.0, 3.0	1.0	WLUAINVALID	Fail
2.0, 3.0	1.0	3.0	1.0	App fails
1.0, 2.0, 3.0	1.0, 2.0, 3.0	3.0	3.0	Use 3.0

Note

The application that uses SLI verbs must call **WinSLIStartup** prior to issuing any other LUA commands. However, **WinSLISt artup** needs to be called only once per application. If it is called multiple times, the subsequent calls will be rejected.

Details of the actual LUA implementation are described in the **WLUADATA** structure, defined as follows:

```
typedef struct { WORD wVersion;
            char szDescription[WLUADESCRIPTION_LEN+1];
            } LUADATA;
```

Having made its last Windows LUA call, an application should call the **WinSLICleanup** routine.

Each LUA application that uses SLI verbs must make a **WinSLIStartup** call before issuing any other LUA calls.

See Also

Reference

WinSLICleanup

SNA Services Enhancement to the Windows LUA Environment

This section describes the Microsoft® Host Integration Server extension to Microsoft Windows® logical unit application (LUA) that converts primary and secondary return codes in the verb control block (VCB) to a printable string.

This section contains:

- GetLuaReturnCode

GetLuaReturnCode

The **GetLuaReturnCode** function converts the primary and secondary return codes in the verb control block (VCB) to a printable string. This function provides a standard set of error strings for use by logical unit application (LUA) applications.

Syntax

```
            int WINAPI GetLuaReturnCode(
    struct LUA_COMMON FAR *vpb,
        UINT buffer_length,
        unsigned char FAR *buffer_addr );
```

Parameters

vpb

Supplied parameter. Specifies the address of the verb control block.

buffer_length

Supplied parameter. Specifies the length of the buffer pointed to by *buffer_addr*. The recommended length is 256.

buffer_addr

Supplied/returned parameter. Specifies the address of the buffer that will hold the formatted, null-terminated string.

Remarks

Return Codes

0x20000001

The parameters are invalid; the function could not read from the specified verb control block or could not write to the specified buffer.

0x20000002

The specified buffer is too small.

0x20000003

The LUA string library LUAST32.DLL could not be loaded.

Remarks

The descriptive error string returned in *buffer_addr* does not terminate with a newline character (**\n**).

The descriptive error strings are contained in LUAST32.DLL and can be customized for different languages.

LUA Verb Control Blocks

When an application issues a Microsoft® Windows® logical unit application (LUA) verb, the verb is coded within the application as a precisely defined verb control block (VCB). The total length of this VCB is variable and is defined by **lua_verb_length**.

This section defines the structure of individual Windows LUA VCBs.

This section contains:

Common Structure of LUA VCBs

The following data structure shows the parameters that are common to all Microsoft® Windows® logical unit application (LUA) verbs.

Syntax

```
struct LUA_COMMON {
    unsigned short lua_verb;
    unsigned short lua_verb_length;
    unsigned short lua_prim_rc;
    unsigned long  lua_sec_rc;
    unsigned short lua_opcode;
    unsigned long  lua_correlator;
    unsigned char  lua_luname[8];
    unsigned short lua_extension_list_offset;
    unsigned short lua_cobol_offset;
    unsigned long  lua_sid;
    unsigned short lua_max_length;
    unsigned short lua_data_length;
    char FAR *     lua_data_ptr;
    unsigned long  lua_post_handle;
    struct LUA_TH  lua_th;
    struct LUA_RH  lua_rh;
    struct LUA_FLAG1 lua_flag1;
    unsigned char  lua_message_type;
    struct LUA_FLAG2 lua_flag2;
    unsigned char  lua_resv56[7];
    unsigned char  lua_encr_decr_option;
} LUA_COMMON;
```

Remarks

Members

lua_verb

Supplied parameter. Contains the verb code, LUA_VERB_RUI for Request Unit Interface (RUI) verbs or LUA_VERB_SLI for SLI verbs. For both of these macros the value is 0x5200.

lua_verb_length

Supplied parameter. Specifies the length in bytes of the LUA VCB. It must contain the length of the verb record being issued.

lua_prim_rc

Primary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_sec_rc

Secondary return code set by LUA at the completion of the verb. The valid return codes vary depending on the LUA verb issued.

lua_opcode

Supplied parameter. Contains the LUA command code (verb operation code) for the verb to be issued, for example, LUA_OPCODE_RUI_BID for the RUI_BID verb. Valid values are as follows:

LUA_OPCODE_SLI_OPEN

LUA_OPCODE_SLI_CLOSE

LUA_OPCODE_SLI_RECEIVE

LUA_OPCODE_SLI_SEND

LUA_OPCODE_SLI_PURGE

LUA_OPCODE_SLI_BID

LUA_OPCODE_SLI_BIND_ROUTINE

LUA_OPCODE_SLI_STSN_ROUTINE

LUA_OPCODE_SLI_CRV_ROUTINE

LUA_OPCODE_RUI_INIT

LUA_OPCODE_RUI_TERM

LUA_OPCODE_RUI_READ

LUA_OPCODE_RUI_WRITE

LUA_OPCODE_RUI_PURGE

LUA_OPCODE_RUI_BID

lua_correlator

Supplied parameter. Contains a user-supplied value that links the verb with other user-supplied information. LUA does not use or change this information. This parameter is optional.

lua_luname

Supplied parameter. Specifies the ASCII name of the local LU used by the Windows LUA session.

SLI_OPEN and RUI_INIT require this parameter. Other Windows LUA verbs only require this parameter if **lua_sid** is zero.

This parameter is eight bytes long, padded on the right with spaces (0x20) if the name is shorter than eight characters.

lua_extension_list_offset

Specifies the offset from the start of the VCB to the extension list of user-supplied dynamic-link libraries (DLLs). This parameter is not used by RUI in Microsoft® Host Integration Server and should be set to zero. The value must be the beginning of a word boundary unless there is no extension list.

lua_cobol_offset

Offset of the COBOL extension. Not used by LUA in Host Integration Server and should be zero.

lua_sid

Supplied and returned parameter. Specifies the session identifier and is returned by SLI_OPEN and RUI_INIT. Other verbs use this parameter to identify the session used for the command. If other verbs use the **lua_luname** parameter to identify sessions, set the **lua_sid** parameter to zero.

lua_max_length

Specifies the length of received buffer for RUI_READand SLI_RECEIVE. For other RUI and SLI verbs, it is not used and should be set to zero.

lua_data_length

Specifies the length of the data being sent or received. It specifies the length of data returned in **lua_peek_data** for the RUI_BID verb.

lua_data_ptr

Pointer to an application-supplied buffer.

When SLI_RECEIVE or RUI_READ is issued, this parameter points to the location to receive the data from the host.

When SLI_SEND or RUI_WRITE is issued, this parameter points to the location of the application's data to be sent to the host.

When SLI_PURGE or RUI_PURGE is issued, this parameter points to the location of the **SLI_RECEIVE** or **RUI_READ** verb's VCB that is to be canceled.

When SLI_OPEN is issued, this parameter can be one of the following:

- The logon message for the SSCP normal flow when the initialization type is secondary with an unformatted logon message.

- The request/response unit (RU) for INITSELF. When the initialization type is secondary with INITSELF, the necessary data for the application is provided.

- For all other open types, this field should be set to zero.

For other RUI and Session Level Interface (SLI) verbs, this parameter is not used and should be set to zero. Both SNA commands and data are placed in this buffer, and they can be in an EBCDIC format.

This information is provided by the Windows LUA application.

lua_post_handle

Supplied parameter. Used under Microsoft Windows® 2000 and Windows Server™ 2003 if asynchronous notification is to be accomplished by events. This variable contains the handle of the event to be signaled or a window handle.

lua_th

Returned parameter. Contains the SNA transmission header (TH) of the message sent or received. Various subparameters are set for write functions and returned for read and bid functions. The subparameters are as follows:

lua_th.flags_fid

Format identification type 2, four bits.

lua_th.flags_mpf

Segmenting mapping field, two bits. Defines the type of data segment. The following values are valid:

0x00 Middle segment**0x04** Last segment**0x08** First segment**0x0C** Only segment

lua_th.flags_odai

Originating address field–destination address field (OAF–DAF) assignor indicator, one bit.

lua_th.flags_efi

Expedited flow indicator, one bit.

lua_th.daf

Destination address field (DAF), an unsigned char.

lua_th.oaf

Originating address field (OAF), an unsigned char.

lua_th.snf

Sequence number field, an unsigned char[2].

lua_rh

Returned parameter. Contains the SNA request/response header (RH) of the message sent or received. It is set for the write function and returned by the read and bid functions. Its sub parameters are as follows:

lua_rh.rri

Request-response indicator, one bit.

lua_rh.ruc

RU category, two bits. The following values are valid:

LUA_RH_FMD (**0x00**) FM data segment**LUA_RH_NC** (**0x20**) Network control**LUA_RH_DFC** (**0x40**) Data flow control**LUA_RH_SC** (**0x60**) Session control

lua_rh.fi

Format indicator, one bit.

lua_rh.sdi

Sense data included indicator, one bit.

lua_rh.bci

Begin chain indicator, one bit.

lua_rh.eci

End chain indicator, one bit.

lua_rh.dr1i

Definite response 1 indicator, one bit.

lua_rh.dr2i

Definite response 2 indicator, one bit.

lua_rh.ri

Exception response indicator (for a request), or response type indicator (for a response), one bit.

lua_rh.qri

Queued response indicator, one bit.

lua_rh.pi

Pacing indicator, one bit.

lua_rh.bbi

Begin bracket indicator, one bit.

lua_rh.ebi

End bracket indicator, one bit.

lua_rh.cdi

Change direction indicator, one bit.

lua_rh.csi

Code selection indicator, one bit.

lua_rh.edi

Enciphered data indicator, one bit.

lua_rh.pdi

Padded data indicator, one bit.

lua_flag1

Supplied parameter. Contains a data structure containing flags for messages supplied by the application. This parameter is used by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, SLI_RECEIVE, and SLI_SEND. For other LUA verbs this parameter is not used and should be set to zero. Its subparameters are as follows:

lua_flag1.bid_enable

Bid enable indicator, one bit.

lua_flag1.close_abend

Close immediate indicator, one bit.

lua_flag1.nowait

No wait for data flag, one bit.

lua_flag1.sscp_exp

SSCP expedited flow, one bit.

lua_flag1.sscp_norm

SSCP normal flow, one bit.

lua_flag1.lu_exp

LU expedited flow, one bit.

lua_flag1.lu_norm

LU normal flow, one bit.

lua_message_type

Specifies the type of the inbound or outbound SNA commands and data. This is a returned parameter for RUI_INIT and SLI_OPEN and a supplied parameter for SLI_SEND. For other LUA verbs this variable is not used and should be set to zero.

Possible values are:

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

The SLI receives and responds to the BIND, CRV, and STSN requests through the LUA interface extension routines.

LU_DATA, LUSTAT_LU, LUSTAT_SSCP, and SSCP_DATA are not SNA commands.

lua_flag2

Returned parameter. Contains flags for messages returned by LUA. This parameter is returned by RUI_BID, RUI_READ, RUI_WRITE, SLI_BID, SLI_RECEIVE, and SLI_SEND. For other LUA verbs this parameter is not used and should be set to zero. Its subparameters are as follows:

lua_flag2.bid_enable

Indicates that **RUI_BID** was successfully re-enabled if set to 1.

lua_flag2.async

Indicates that the LUA interface verb completed asynchronously if set to 1.

lua_flag2.sscp_exp

Indicates SSCP expedited flow if set to 1.

lua_flag2.sscp_norm

Indicates SSCP normal flow if set to 1.

lua_flag2.lu_exp

Indicates LU expedited flow if set to 1.

lua_flag2.lu_norm

Indicates LU normal flow if set to 1.

lua_resv56

This supplied parameter is a reserved field used by SLI_OPEN and RUI_INIT. For all other LUA verbs, this parameter is reserved and should be set to zero.

lua_encr_decr_option

This parameter is a field for cryptography options. On **RUI_INIT**, only the following are supported:

- **lua_encr_decr_option** = 0
- **lua_encr_decr_option** = 128

For all other LUA verbs, this parameter is reserved and should be set to zero.

Values for lua_message_type

The following table describes the possible values for **lua_message_type**.

Message type	SNA data	SLI_SEND	SLI_BID SLI_RECEIVE	RUI_BID RUI_READ
0xC8	BID	X	X	X
0x31	BIND		Extension*	X
0x70	BIS	X	X	X
0x83	CANCEL	X	X	X
0x84	CHASE	X	X	X
0xA1	CLEAR			X
0xD0	CRV			X
0x01	LU_DATA**	X	X	X
0x04	LUSTAT_LU**	X	X	X
0x14	LUSTAT_SSCP**	X	X	X
0x81	QC	X	X	X
0x80	QEC	X	X	X
0x82	RELQ	X	X	X
0xA3	RQR	X		X
0x02	RSP	X	X	
0x05	RTR	X	X	X
0x71	SBI	X	X	X
0xC0	SHUTD			X
0xC9	SIGNAL	X	X	X
0xA0	SDT			X
0x11	SSCP_DATA**	X	X	X
0xA2	STSN		Extension*	X
0x32	UNBIND			X

*The SLI receives and responds to the BIND, CRV, and STSN requests through the LUA interface extension routines.

**Not an SNA command.

Command-Specific Structure of LUA VCBs

The following union shows the specific data structure that is included for functions that use the **LUA_SPECIFIC** part of a verb control block. The only logical unit application (LUA) verbs that use this union are RUI_BID,SLI_BID,SLI_OPEN, and SLI_SEND.

Syntax

```
union LUA_SPECIFIC {
    struct SLI_OPEN open;
    unsigned char lua_sequence_number[2];
    unsigned char lua_peek_data[12];
} LUA_SPECIFIC;
```

Remarks

Members

open

The union member of **LUA_SPECIFIC** used by the **SLI_OPEN** verb.

lua_sequence_number

The union member of **LUA_SPECIFIC** used by the **SLI_SEND** verb. Returned parameter. Sequence number of the RU to the host. It contains the sequence number for either the first in the chain request unit or the only segment in the chain request unit. Note that this parameter is not byte-reversed.

lua_peek_data

The union member of **LUA_SPECIFIC** used by the **RUI_BID** and **SLI_BID** verbs. Returned parameter. Contains up to 12 bytes of the data waiting to be read. It is a preview (up to 12 bytes) of the request/response unit (RU) data waiting to be read. The **lua_data_length** parameter contains the exact length of the data peeked at.

The following topic describes command-specific parameters for **SLI_OPEN**.

This section contains:

- SLI_OPEN VCB Structure

SLI_OPEN VCB Structure

The following structure shows the **SLI_OPEN** fields of the **LUA SPECIFIC** union member for the SLI_OPEN verb.

Syntax

```
struct SLI_OPEN {
    unsigned char lua_init_type;
    unsigned char lua_resv65;
    unsigned short lua_wait;
    struct LUA_EXT_ENTRY lua_open_extension[3];
    unsigned char lua_ending_delim;
} SLI_OPEN;
```

Remarks

Members

lua_init_type

Type of session initiation, which determines how the LU-LU session is initialized by the Windows LUA interface. The following values are valid:

lua_init_type_sec_is

Secondary-initiated and sends the INITSELF command supplied in the OPEN data buffer.

lua_init_type_sec_log

Secondary-initiated with an unformatted LOGON message in the OPEN data buffer.

lua_init_type_prim

Primary-initiated and waits on the BIND command.

lua_init_type_prim_sscp

Primary-initiated with SSCP access.

lua_resv65

Reserved field.

lua_wait

Secondary retry wait time. Specifies how many seconds the Windows LUA interface is to wait before retransmitting the INITSELF or the LOGON message after receiving one of the following:

- A NOTIFY command (indicating a procedure error)
- A network services procedure error message
- A negative response with one of the following secondary return codes:

  RESOURCE_NOT_AVAILABLE SESSION_LIMIT_EXCEEDED SESSION_SERVICE_PATH_ERROR

lua_open_extension

Supplied parameter. Specifies any user-supplied dynamic-link libraries (DLLs) used to process specific LUA messages.

lua_ending_delim

Extension list delimiter.

LUA Common Return Codes

This section describes the primary and, if applicable, secondary return codes that are common to the logical unit application (LUA) verbs. The return codes are listed in hexadecimal order.

Verb-specific return codes are described for the individual verbs in LUA RUI Verbs and LUA SLI Verbs.

This section contains:

LUA Primary Return Codes

0x0000
LUA_OK

The verb executed successfully.

0x0001
LUA_PARAMETER_CHECK

The verb did not execute because of a parameter error.

0x0002
LUA_STATE_CHECK

The verb did not execute because it was issued in an invalid state.

0x000F
LUA_SESSION_FAILURE

A required Microsoft® Host Integration Server component (such as the local node) has terminated.

0x0014
LUA_UNSUCCESSFUL

The verb record supplied was valid, but the verb did not complete successfully.

0x0018
LUA_NEGATIVE_RESPONSE

Either the logical unit application (LUA) sent a negative response to a message received from the primary logical unit (PLU) because an error was found in the message, or the application responded negatively to a chain for which the end-of-chain has arrived.

0x0021
LUA_CANCELED

The secondary return code gives the reason for canceling the command.

0x0030
LUA_IN_PROGRESS

An asynchronous command was received but is not completed.

0x0040
LUA_STATUS

The secondary return code contains Session Level Interface (SLI) status information for the application.

0xF003
LUA_COMM_SUBSYSTEM_ABENDED

Indicates one of the following conditions:

- The node used by this conversation encountered an ABEND.
- The connection between the transaction program (TP) and the physical unit (PU) 2.1 node was broken (a LAN error).
- The SnaBase at the TPs computer encountered an ABEND.

0xF004
LUA_COMM_SUBSYSTEM_NOT_LOADED

A required component could not be loaded or terminated while processing the verb. Thus, communication could not take place. Contact the system administrator for corrective action.

0xF008

LUA_INVALID_VERB_SEGMENT

The verb control block (VCB) extended beyond the end of the data segment.

0xF011
LUA_UNEXPECTED_DOS_ERROR

After issuing an operating system call, an unexpected operating system return code was received and is specified in the secondary return code.

0xF015
LUA_STACK_TOO_SMALL

The stack size of the application is too small to execute the verb. Increase the stack size of your application.

0xFFFF
LUA_INVALID_VERB

Either the verb code or the operation code, or both, is invalid. The verb did not execute.

LUA Secondary Return Codes

0x00000000
LUA_SEC_RC_OK

No additional information exists for LUA_OK.

0x00000001
LUA_INVALID_LUNAME

An invalid **lua_luname** name was specified.

0x00000002
LUA_BAD_SESSION_ID

An invalid value for **lua_sid** was specified in the verb control block (VCB).

0x00000003
LUA_DATA_TRUNCATED

The data was truncated because the data received was longer than the buffer length specified in **lua_max_length**.

0x00000004
LUA_BAD_DATA_PTR

The **lua_data_ptr** parameter either does not contain a valid pointer or does not point to a read/write segment, and supplied data is required.

0x00000005
LUA_DATA_LENGTH_ERROR

One of the following occurred:

- The supplied data segment for **SLI_RECEIVE** or **SLI_SEND** is not a read/write data segment as required.
- The supplied data segment for **SLI_RECEIVE** is not as long as that provided in **lua_max_length**.
- The supplied data segment for **SLI_SEND** is not as long as that provided in **lua_data_length**.

0x00000006
LUA_RESERVED_FIELD_NOT_ZERO

A reserved parameter for the verb just issued is not set to zero.

0x00000007
LUA_INVALID_POST_HANDLE

For a Microsoft Windows Server 2003 or Windows 2000 system using events as the asynchronous posting method, the Windows-based logical unit application (LUA) VCB does not contain a valid event handle.

0x0000000C
LUA_PURGED

SLI_PURGE was issued and canceled **SLI_RECEIVE**.

0x0000000F
LUA_BID_VERB_ERROR

The buffer with the **SLI_BID** VCB was released before the **SLI_RECEIVE** with **lua_flag1.bid_enable** set to 1 was issued.

0x00000010
LUA_NO_PREVIOUS_BID_ENABLED

SLI_BID was not issued prior to issuing **SLI_RECEIVE** with **bid_enable**.

0x00000011
LUA_NO_DATA

No data was available to read when **SLI_RECEIVE** containing a no-wait parameter was issued.

0x00000012
LUA_BID_ALREADY_ENABLED

SLI_RECEIVE was issued with **bid_enable** when **SLI_BID** was already active.

0x00000013
LUA_VERB_RECORD_SPANS_SEGMENTS

The LUA VCB length parameter plus the segment offset is beyond the segment end.

0x00000014
LUA_INVALID_FLOW

The **lua_flag1** flow flags were set incorrectly when a verb was issued as follows:

- When issuing **SLI_SEND** to send an SNA response, set only one **lua_flag1** flow flag.
- When issuing **SLI_RECEIVE**, set at least one **lua_flag1** flow flag.

0x00000015
LUA_NOT_ACTIVE

LUA was not active within Microsoft Host Integration Server when an LUA verb was issued.

0x00000016
LUA_VERB_LENGTH_INVALID

An LUA verb was issued with the value of **lua_verb_length** unexpected by the LUA.

0x00000019
LUA_REQUIRED_FIELD_MISSING

The verb that was issued either did not include a data pointer (if the data count was not zero) or did not include an **lua_flag1** flow flag.

0x00000030
LUA_READY

Following a NOT_READY status, this status is issued to notify you that the Session Level Interface (SLI) is ready to process commands.

0x00000031
LUA_NOT_READY

One of the following caused the SLI session to be temporarily suspended:

- An SNA UNBIND type 0x02 command was received, indicating a new BIND is coming.

  If the UNBIND type 0x02 is received after the beginning **SLI_OPEN** is complete, the session is suspended until a BIND, optional CRV and STSN, and SDT flows are received. These routines are re-entrant because they must be called again. The session resumes after the SLI processes the SDT command. If the UNBIND type 0x02 is received while **SLI_OPEN** is still processing, the primary return code is session-failure, not status.

- The receipt of an SNA CLEAR caused the suspension.

  Receipt of an SNA SDT will cause the session to resume.

0x00000032
LUA_INIT_COMPLETE

The LUA interface initialized the session while **SLI_OPEN** was processing. LUA applications that issue **SLI_OPEN** with **lua_open_type_prim_sscp** receive this status on **SLI_RECEIVE** or **SLI_BID**.

0x00000033
LUA_SESSION_END_REQUESTED

The LUA interface received an SNA shutdown command (SHUTD) from the host, indicating the host is ready to shut down the session.

0x00000034
LUA_NO_SLI_SESSION

A session was not open or was down due to an **SLI_CLOSE** or session failure when a command was issued.

0x00000035
LUA_SESSION_ALREADY_OPEN

A session is already open for the logical unit (LU) name specified in **SLI_OPEN**.

0x00000036
LUA_INVALID_OPEN_INIT_TYPE

The value in the **lua_init_type** contained in **SLI_OPEN** is invalid.

0x00000037
LUA_INVALID_OPEN_DATA

The **lua_init_type** for the **SLI_OPEN** issued is set to LUA_INIT_TYPE_SEC_IS when the buffer for data does not have a valid INITSELF command.

0x00000038
LUA_UNEXPECTED_SNA_SEQUENCE

Unexpected data or commands were received from the host while **SLI_OPEN** was processing.

0x00000039
LUA_NEG_RSP_FROM_BIND_ROUTINE

The user-supplied SLI_BIND routine responded negatively to the BIND. **SLI_OPEN** ended unsuccessfully.

0x0000003B
LUA_NEG_RSP_FROM_STSN_ROUTINE

The user-supplied SLI STSN routine responded negatively to the STSN. **SLI_OPEN** ended unsuccessfully.

0x0000003E
LUA_INVALID_OPEN_ROUTINE_TYPE

The **lua_open_routine_type** for the **SLI_OPEN** list of extension routines is invalid.

0x0000003F
LUA_MAX_NUMBER_OF_SENDS

The application issued a third **SLI_SEND** before one completed.

0x00000040
LUA_SEND_ON_FLOW_PENDING

An **SLI_SEND** was still outstanding when the application issued another **SLI_SEND** for an SNA flow.

0x00000041
LUA_INVALID_MESSAGE_TYPE

The **lua_message_type** parameter is not recognized by the LUA interface.

0x00000042
LUA_RECEIVE_ON_FLOW_PENDING

An **SLI_RECEIVE** was still outstanding when this application issued another **SLI_RECEIVE** for an SNA flow.

0x00000043
LUA_DATA_LENGTH_ERROR

The application did not provide user-supplied data required by the verb issued. Note that when **SLI_SEND** is issued for an SNA LUSTAT command, status (in four bytes) is required, and that when **SLI_OPEN** is issued with secondary initialization, data is required.

0x00000044
LUA_CLOSE_PENDING

One of the following occurred:

- A CLOSE_ABEND was still pending when another CLOSE_ABEND was issued. You can issue a CLOSE_ABEND if a CLOSE_NORMAL is pending.
- Either a CLOSE_ABEND or a CLOSE_NORMAL was still pending when a CLOSE_NORMAL was issued.

0x00000046
LUA_NEGATIVE_RSP_CHASE

A negative response to an SNA CHASE command from the host was received by the LUA interface while **SLI_CLOSE** was being processed. **SLI_CLOSE** continued processing to stop the session.

0x00000047
LUA_NEGATIVE_RSP_SHUTC

A negative response to an SNA SHUTC command from the host was received by the SLI while **SLI_CLOSE** was still being processed. **SLI_CLOSE** continued processing to stop the session.

0x00000048
LUA_NEGATIVE_RSP_RSHUTD

A negative response to an SNA RSHUTD command from the host was received by the LUA interface while **SLI_CLOSE** was being processed. **SLI_CLOSE** continued processing to stop the session.

0x0000004A
LUA_NO_RECEIVE_TO_PURGE

No **SLI_RECEIVE** was outstanding when you issued **SLI_PURGE**. One of two situations caused the problem:

- **SLI_RECEIVE** completed before **SLI_PURGE** finished processing.

  You can change the application to take care of this problem because it is not an error condition.

- The **lua_data_ptr** parameter does not correctly point to the **SLI_RECEIVE** you want to purge.

0x0000004D
LUA_CANCEL_COMMAND_RECEIVED

The host sent an SNA CANCEL command to cancel the data chain currently being received by **SLI_RECEIVE**.

0x0000004E
LUA_RUI_WRITE_FAILURE

An unexpected error was posted to the SLI by **RUI_WRITE**.

0x00000051
LUA_SLI_BID_PENDING

An SLI verb was still active when another **SLI_BID** was issued. Only one **SLI_BID** can be active at a time.

0x00000052
LUA_SLI_PURGE_PENDING

An **SLI_PURGE** was still active when another **SLI_PURGE** was issued. Only one **SLI_PURGE** can be active at a time.

0x00000053
LUA_PROCEDURE_ERROR

A host procedure error is indicated by the receipt of an NSPE or NOTIFY message. The return code is posted to **SLI_OPEN** when the retry option is not used. To use the reset option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue **SLI_CLOSE**.

0x00000054
LUA_INVALID_SLI_ENCR_OPTION

The **lua_encr_decr_option** parameter was set to 128 in **SLI_OPEN**, which is not supported for the encryption/decryption processing option.

0x00000055
LUA_RECEIVED_UNBIND

The primary LU sent an SNA UNBIND command to the LUA interface when a session was active. As a result, the session was stopped.

0x0000007F
LUA_SLI_LOGIC_ERROR

The LUA interface found an internal error in logic.

0x00000080
LUA_TERMINATED

The session was terminated when a verb was pending. The verb process has been canceled.

0x00000081
LUA_NO_RUI_SESSION

No session has been initialized for the LUA verb issued, or some verb other than **SLI_OPEN** was issued before the session was initialized.

0x00000083
LUA_INVALID_PROCESS

The session for which a Request Unit Interface (RUI) verb was issued is unavailable because another process owns the session.

0x0000008C
LUA_LINK_NOT_STARTED

The LUA was not able to activate the data link during initialization of the session.

0x0000008D
LUA_INVALID_ADAPTER

The configuration for the data link control (DLC) is in error, or the configuration file is corrupted.

0x0000008E
LUA_ENCR_DECR_LOAD_ERROR

An unexpected return code was received from the OS/2 **DosLoadModule** function while attempting to load the user-provided encryption or decryption dynamic link module.

0x0000008F
LUA_ENCR_DECR_LOAD_ERROR

An unexpected return code was received from the OS/2 **DosGetProcAddr** function while attempting to get the procedure address within the user-provided encryption or decryption dynamic link module.

0x000000BE
LUA_NEG_NOTIFY_RSP

The system services control point (SSCP) responded negatively to a NOTIFY request issued indicating that the secondary LU was capable of a session. The half-session component that received the request understood and supported the request but could not execute it.

0x000000FF
LUA_LU_INOPERATIVE

A severe error occurred while the RUI was attempting to stop the session. This LU is unavailable for any LUA requests until an ACTLU is received from the host.

0x08010000
LUA_RESOURCE_NOT_AVAILABLE

The logical unit, physical unit, link, or link station specified in the request unit is unavailable. This return code is posted to **SLI_OPEN** when a resource is unavailable unless you use the retry option.

To use the retry option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue **SLI_CLOSE**.

0x08050000

LUA_SESSION_LIMIT_EXCEEDED

The session requested was not activated because a network addressable unit (NAU) is at its session limit.

This SNA sense code applies to the following requests: BID, CINIT, INIT, and ACTDRM. The code will be posted to **SLI_OPEN** when an NAU is at its limit, unless you use the retry option.

To use the retry option, set **lua_wait** to a value other than zero. The LOGON or INITSELF command will be retried until the host is ready or until you issue **SLI_CLOSE**.

0x08090000
LUA_MODE_INCONSISTENCY

Performing this function is not allowed by the current status. The request sent to the half-session component was not executed even though it was understood and supported. This SNA sense code is also an exception request sense code.

0x08120000
LUA_INSUFFICIENT_RESOURCES

A temporary condition of insufficient resources caused the request receiver to be unable to perform. The request sent to the half-session component was not executed, even though it was understood and supported.

0x081B0000
LUA_RECEIVER_IN_TRANSMIT_MODE

Either resources needed to handle normal flow data were not available or the state of the half-duplex contention was not received when a normal-flow request was received. The result is a race condition. This SNA sense code is also an exception request sense code.

0x08310000
LUA_LU_COMPONENT_DISCONNECTED

An LU component is unavailable because it is not connected properly. Make sure that the power is on.

0x08350001
LUA_NEGOTIABLE_BIND_ERROR

A negotiable BIND was received, which is only allowed by the SLI when a user-supplied SLI_BIND routine is provided with **SLI_OPEN**.

0x08350002
LUA_BIND_FM_PROFILE_ERROR

Only file management header profiles 3 and 4 are supported by the LUA interface. A file management profile other than 3 or 4 was found on the BIND.

0x08350003
LUA_BIND_TS_PROFILE_ERROR

Only Transmission Service (TS) profiles 3 and 4 are supported by the LUA interface. A TS profile other than 3 or 4 was found on the BIND.

0x0835000E
LUA_BIND_LU_TYPE_ERROR

Only LU 0, LU 1, LU 2, and LU 3 are supported by LUA. An LU other than 0, 1, 2, or 3 was found.

0x08570000
LUA_SSCP_LU_SESSION_NOT_ACTIVE

The required SSCP-LU is inactive. Specific sense code information is in bytes 2 and 3. Valid settings are 0x0000, 0x0001, 0x0002, 0x0003, and 0x0004.

0x08780001
LUA_RECEIVE_CORRELATION_TABLE_FULL

The session receive correlation table for the flow requested reached its capacity.

0x08780002
LUA_SEND_CORR_TABLE_FULL

The session send correlation table for the flow requested reached its capacity.

0x087D0000
LUA_SESSION_SERVICE_PATH_ERROR

A request for session services cannot be rerouted to an SSCP-SSCP session path. Specific sense code information in bytes 2 and 3 gives more information about why the request cannot be rerouted.

0x10020000
LUA_RU_LENGTH_ERROR

The request/response unit (RU) request was an incorrect length (either too short or too long). The RU was not interpreted or processed even though it was delivered to the half-session component. The half-session capabilities do not match. This SNA sense code is also an exception request sense code.

0x10030000
LUA_FUNCTION_NOT_SUPPORTED

The LUA does not support the requested function. A control character, an RU parameter, or a formatted request code may have specified the function. Specific sense code information is in bytes 2 and 3.

0x10050121
LUA_HDX_BRACKET_STATE_ERROR

The existing state error prevented the current request from being sent. The determination was made by a protocol computer.

0x10050122
LUA_RESPONSE_ALREADY_SENT

A response for the chain was already sent so that the current request was not sent. The determination was made by a protocol computer.

0x10050123
LUA_EXR_SENSE_INCORRECT

The application responded negatively to an exception request. The sense code was unacceptable.

0x10050124
LUA_RESPONSE_OUT_OF_ORDER

The current response was not for the oldest request. The determination was made by a protocol computer.

0x10050125
LUA_CHASE_RESPONSE_REQUIRED

A CHASE response was still outstanding when a more recent request was attempted. The determination was made by a protocol computer.

0x20020000
LUA_CHAINING_ERROR

The sequence of the chain indicator settings is in error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20030000
LUA_BRACKET

The sender failed to enforce the session bracket rules. Note that contention and race conditions are exempt from this error. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20040000
LUA_DIRECTION

While the half-duplex flip-flop state was NOT_RECEIVE, a request for normal flow was received. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20050000
LUA_DATA_TRAFFIC_RESET

A half-session of an active session with inactive data traffic received a normal flow DFC or FMD request. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20060000
LUA_DATA_TRAFFIC_QUIESCED

A data flow control (DFC) or function management data (FMD) request was received from a half-session that sent either a SHUTC command or QC command, and the DFC or FMD request has not responded to a RELQ command. An invalid request header or request unit for the receivers current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20070000
LUA_DATA_TRAFFIC_NOT_RESET

While the data traffic state was not reset, the session control request was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20080000
LUA_NO_BEGIN_BRACKET

The receiver has already sent a positive response to a Bracket Initiation Stopped (BIS) command when a BID or an FMD request specifying BBI=BB was received. An invalid request header or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x20090000
LUA_SC_PROTOCOL_VIOLATION

A violation of the session control (SC) protocol occurred. A request (that is permitted only after an SC request and a positive response to that request have been successfully exchanged) was received before the required exchange. Byte 4 of the sense data contains the request code. No user data exists for this sense code. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x200A0000
LUA_IMMEDIATE_REQUEST_MODE_ERROR

The request violated the immediate request mode protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x200B0000
LUA_QUEUED_RESPONSE_ERROR

The request violated the queued response protocol. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x200C0000
LUA_ERP_SYNC_EVENT_ERROR

A violation of the ERP synchronous event protocol occurred. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x200D0000
LUA_RSP_BEFORE_SENDING_REQ

A previously received request has not yet been responded to and an attempt was made in half-duplex send/receive mode to send a normal flow request. An invalid header request or request unit for the received current session control or data flow control state was found. Delivery to the half-session component was prevented.

0x200E0000
LUA_RSP_CORRELATION_ERROR

A response was sent that does not correspond to a previously received request or a response was received that does not correspond to a request sent previously.

0x200F0000
LUA_RSP_PROTOCOL_ERROR

A violation of the response protocol was found in the response received from the primary half-session.

0x40010000
LUA_INVALID_SC_OR_NC_RH

The request/response header (RH) of a session control (SC) or network control (NC) request was invalid.

0x40030000
LUA_BB_NOT_ALLOWED

The begin bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40040000
LUA_EB_NOT_ALLOWED

The end bracket indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40060000
LUA_EXCEPTION_RSP_NOT_ALLOWED

When an exception response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40070000
LUA_DEFINITE_RSP_NOT_ALLOWED

When a definite response was not allowed, one was requested. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40080000
LUA_PACING_NOT_SUPPORTED

The request contained a pacing indicator when support of pacing for this session does not exist for the receiving half-session or boundary function half-session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40090000
LUA_CD_NOT_ALLOWED

The change-direction indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x400A0000
LUA_NO_RESPONSE_NOT_ALLOWED

A request other than an EXR contained a NO RESPONSE. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x400B0000
LUA_CHAINING_NOT_SUPPORTED

The chaining indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x400C0000
LUA_BRACKETS_NOT_SUPPORTED

The bracket indicators were incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x400D0000
LUA_CD_NOT_SUPPORTED

The change-direction indicator was set, but LUA does not support change-direction for this situation. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-

session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x400F0000
LUA_INCORRECT_USE_OF_FI

The format indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40100000
LUA_ALTERNATE_CODE_NOT_SUPPORTED

The code selection indicator was set, but LUA does not support code selection for this session. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40110000
LUA_INCORRECT_RU_CATEGORY

The request unit category indicator was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40120000
LUA_INCORRECT_REQUEST_CODE

The request code was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40130000
LUA_INCORRECT_SPEC_OF_SDI_RTI

The sense-data-included indicator (SDI) and the response-type indicator (RTI) were not specified correctly on a response. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40140000
LUA_INCORRECT_DR1I_DR2I_ERI

The DR1I, the DR2I, and the exception response indicator (ERI) were specified incorrectly. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40150000
LUA_INCORRECT_USE_OF QRI

The queued response indicator (QRI) was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40160000
LUA_INCORRECT_USE_OF_EDI

The EDI was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x40170000
LUA_INCORRECT_USE_OF_PDI

The padded data indicator (PDI) was incorrectly specified. The BIND options chosen previously or the architectural rules were violated by the request header parameter values. Delivery to the half-session component was prevented. The errors are not dependent on the current session state. The senders failure to enforce session rules may have caused the errors.

0x80030000

LUA_NAU_INOPERATIVE

The network addressable unit (NAU) is not able to process responses or requests. Delivery to the receiver could not take place for one of the following reasons:

- A path information unit error
- A path outage
- An invalid sequence of requests for activation

If a path error is received during an active session that usually means there is no longer a valid path to the session partner.

0x80050000
LUA_NO_SESSION

A request to activate a session is required because no active half-session in the receiving end node for the origination-destination pair exists, or no active boundary function half-session component for the origination-destination pair in a node that supplies the boundary function exists. Delivery of the request could not take place for one of the following reasons:

- A path information unit error
- A path outage
- An invalid sequence of requests for activation

If a path error is received during an active session that usually indicates there is no longer a valid path to the session partner.

3270 Emulation Programmer's Reference

This section contains reference material for the 3270 Emulator.

For general information about programming for the 3270 Emulator, see the 3270 Emulation Programmer's Guide section of the SDK.

In This Section

DL-BASE/DMOD Entry Points

FMI Message Formats

FMI Extension for the Windows Environment

DL-BASE/DMOD Entry Points

This section provides definitions for the entry points to the DL-BASE and the Dynamic Access Module (DMOD).

In This Section

- CMDGoTSR
- CMDSemClear
- CMDSemRequest
- CMDSemSet
- CMDSemWait
- CMDStartFG
- CMDStopFG
- RegisterSwitchProc
- routproc
- sbpibegt
- sbpiberl
- sbpuinit
- sbpurcvx
- sbpusend
- sbputerm
- sepdbubl
- sepdburl
- sepdchnk
- sepdcrec
- sepdgetinfo
- sepdrout
- sepwrout

CMDGoTSR

The **CMDGoTSR** function initiates a background thread for the emulator, and then executes an MS-DOS® terminate-and-stay-resident (TSR) interrupt.

MS-DOS

Syntax

```
SHORT APIENTRY CMDGoTSR(
ULONG
            entryPoint,
            UCHAR FAR
            *stack,
            UCHAR FAR
            *topOfRam
);
```

Parameters

entryPoint

Pointer to the function where the background thread will start execution.

stack

Pointer to the stack of the background thread.

topOfRam

Top of RAM; all memory above this address will be released by the Network Access Program (NAP) for LAN Manager or the NAP for NetWare (LMBASE and NWBASE, respectively).

Remarks

An emulator should complete its initialization, and then execute this call to go resident. A thread of execution will be created at the entry point specified.

This call will never return control to the calling program.

CMDSemClear

The **CMDSemClear** function clears a RAM semaphore.

MS-DOS

Syntax

```
USHORT FAR CMDSemClear(
ULONG FAR *ramSem
);
```

Parameters

ramSem

Address of the semaphore.

CMDSemRequest

The **CMDSemRequest** function requests a RAM semaphore.

Syntax

```
USHORT FAR CMDSemRequest(
ULONG FAR
*ramSem,
ULONG timeOut
);
```

Parameters

ramSem

Address of the semaphore.

timeOut

Length of time in milliseconds to wait before returning.

Return Value

0

OK.

ERROR_SEM_TIMEOUT

Time-out expired before semaphore operation completed.

ERROR_SEM_OWNED

This thread or another thread owns the semaphore, and the calling thread specified zero time-out.

CMDSemSet

The **CMDSemSet** function sets a RAM semaphore.

Syntax

```
USHORT FAR CMDSemSet(
ULONG FAR *ramSem
);
```

Parameters

ramSem

Address of the semaphore.

CMDSemWait

The **CMDSemWait** function waits until a RAM semaphore is cleared.

Syntax

```
USHORT FAR CMDSemWait(
ULONG FAR *ramSem,
ULONG timeOut
);
```

Parameters

ramSem

Address of the semaphore.

timeOut

Length of time in milliseconds to wait before returning.

Return Value

0

OK.

ERROR_SEM_TIMEOUT

Time-out expired before semaphore operation completed.

ERROR_SEM_OWNED

This thread or another thread owns the semaphore, and the calling thread specified zero time-out.

CMDStartFG

The **CMDStartFG** function requests that scheduling of the foreground thread be resumed.

Syntax

```
USHORT FAR CMDStartFG();
```

Return Value

0

The foreground thread was successfully resumed.

nonzero

The foreground thread could not be resumed.

Remarks

The emulator should issue this call after restoring the screen contents when returning to background operation.

CMDStopFG

The **CMDStopFG** function requests that the foreground thread be suspended.

Syntax

```
USHORT FAR CMDStopFG(
USHORT timeOut
);
```

Parameters

timeOut

Maximum time to wait for the foreground thread to return before stopping it.

Return Value

0

The foreground thread was successfully stopped.

nonzero

The foreground thread was stopped within MS-DOS.

Remarks

The emulator should issue this call when it wishes to enter the foreground. If the return value is nonzero, the foreground thread has been stopped within MS-DOS and it is not safe for the emulator to come to the foreground. Under these circumstances, the emulator should restart the foreground thread by calling CMDStartFG.

If the call was successful, the emulator should save the contents of the screen before writing to it. When returning to background operation, the emulator must restore the screen and call **CMDStartFG** to allow the previous foreground application to continue.

RegisterSwitchProc

The **RegisterSwitchProc** function registers an application procedure that will be called whenever the 3270 emulator is about to be switched in or out of memory by the MS-DOS version 5, MS-DOS version 6, or Windows 3.*x* task-switching code.

MS-DOS

Syntax

```
USHORT FAR RegisterSwitchProc(
ULONG switchProc
);
```

```
VOID FAR PASCAL SwitchProc(
USHORT inOut
);
```

Parameters

inOut

Zero if emulator is about to be switched out of memory, 1 if emulator is about to be switched back into memory.

switchProc

A far pointer to the function where task activity will be notified. The function is defined as follows:

routproc

The **routproc** function is a sample routing procedure. It must be supplied as part of the application. It is called by the Dynamic Access Module (DMOD) with a message that may or may not be for this application The DMOD calls routing procedures in turn until one accepts the message.

Syntax

```
DWORD routproc(
BUFHDR *msgptr,
USHORT locl,
USHORT retstat
);
```

Parameters

msgptr

Pointer to the message passed by the DMOD to the routing procedure.

locl

Locality from which message was received (if *retstat* indicates message returned), or locality to which path was lost (if *retstat* indicates path error).

retstat

Reason for call:

CEDINMSG (1)—message returned.

CEDINLLN (2)—path error (see Remarks below).

Return Value

TRUE

The routing procedure has accepted the message.

FALSE

The message is not for this routing procedure.

Remarks

The routing procedure should first call sbpurcvx, which handles any Open response messages, as follows:

sbpurcvx(&*msgptr*, *locl*, *retstat***)**

A return code of TRUE from **sbpurcvx** indicates that **sbpurcvx** has accepted the message; an Open error response has been received for this application, and resource location is continuing. The routing procedure should not process the message any further and should return **TRUE** to prevent the DMOD from calling further routing procedures.

A return code of **FALSE** from **sbpurcvx** indicates that the routing procedure should:

- If the message is for this application, take responsibility for the message and return **TRUE** to prevent the DMOD from calling further routing procedures.
- If the message is not for this application, return **FALSE** so that the DMOD tries further routing procedures.

If a path error is returned, *msgptr* will not point to a valid message, and no more function management interface (FMI) messages will be returned for the locality value indicated. The application is responsible for ending all sessions using this locality. The routing procedure must return **FALSE**. This ensures that the lost locality is reported to all other routing procedures.

If the message is for this application, the routing procedure can either process the message immediately or put the message on an application queue, and then post the application using a semaphore. For more information, see Receiving Messages.

sbpibegt

The application calls the **sbpibegt** function to get a buffer element to append to an existing buffer.

Syntax

```
VOID sbpibegt(
PTRBFELT *eltptr
);
```

Parameters

eltptr

Pointer to a pointer to an element. On return, this points to a pointer to the element obtained, or to NULL if an element was not obtained (an internal error).

Remarks

This function should only be used to get extra elements for an existing buffer. The sepdbubl function should be used to get a new buffer.

The new element should be added to the chain of elements from the existing buffer header and the count of the number of elements updated.

This function is typically used when a received buffer is being reused to transmit a message that is longer than the incoming message.

sbpiberl

The application calls the **sbpiberl** function to release a buffer element from an existing buffer.

Syntax

```
VOID sbpiberl(
PTRBFELT *eltptr
);
```

Parameters

eltptr

Pointer to a pointer to the element to be released.

Remarks

This function should only be used to release surplus elements from a buffer. The sepdburl function should be called to release the entire buffer.

The released element should first be removed from the element chain and the count of the number of elements updated.

This function is typically used when a received buffer is being reused to transmit a message that is shorter than the incoming message.

sbpuinit

The **sbpuinit** function initializes the DL-BASE.

Syntax

```
USHORT sbpuinit(
HANDLE *sema4ptr,
USHORT proctype,
USHORT servtype,
UCHAR *uname
);
```

Parameters

sema4ptr

Semaphore, created by Dynamic Access Module (DMOD), cleared by DMOD when a message is available. For MS-DOS, the application should supply the address of a 4-byte (long) integer. This address is for internal use by Microsoft Host Integration Server 2009—the application should not subsequently attempt to reference the address.

proctype

Type of process: CLIENT–2.

servtype

Type of service/client: CES3270–2.

uname

Pointer to a character buffer of length at least 21 characters; the LAN Manager user name, or other identifying name appropriate to the network operating system, is returned to the application in this buffer. The application does not need to use this parameter, but can use it for display or logging.

Return Value

NO_ERROR

Initialization successful.

Any other return value indicates that the initialization failed. This is usually an operating system return code. The following values are also used:

DMLTABF (555)

L table is full.

DMMNWGI (562)

Failed to get network operating system information.

DMDSTFL (563)

Service table is full.

DMMPIPF (567)

Failed to make a named pipe.

DMCOMNM (582)

No name registered for this application.

DMCOMDUP (596)

A service is already running with the same name.

DMNOTLOG (598)

User is not logged on to network operating system.

DMCFGOPN (616)

Failed to open configuration file.

DMCFGREAD (618)

Failed to read from configuration file.

DMNONAP (625)

The Network Access Program (NAP) is not started.

DMMAXAPP (953)

Windows only: Maximum number of concurrent applications exceeded.

Remarks

The **sbpuinit** entry point should always be called before any other DL-BASE or DMOD entry points except SNAGetVersion. For new emulators, sepdcrec should be called after **sbpuinit**. (Because of the order of calls used in older emulators, a call to **sepdcrec** before **sbpuinit** is still supported, but this order is not recommended.)

sbpurcvx

The **sbpurcvx** function processes Open responses from a routing procedure. An application can define a routing procedure that is called by the Dynamic Access Module (DMOD) when a message is received. This routing procedure should first call **sbpurcvx** to handle any Open response messages received. This ensures that Open responses intended for the Resource Locator are handled correctly.

Syntax

```
USHORT sbpurcvx(
BUFHDR * *msgptr,
INTEGER locl,
INTEGER retstat
);
```

Parameters

msgptr

Pointer to the message returned by the DMOD to the routing procedure.

locl

Locality from which message was received (if *retstat* indicates message returned), or locality to which path was lost (if *retstat* indicates path error).

retstat

Reason for call:

CEDINMSG (1—message returned.

CEDINLLN (2)—path error.

Return Value

TRUE

The Resource Locator has accepted the message; the application should not process it any further.

FALSE

The message should be processed by the application.

Remarks

This function is called by a routing procedure that is called by the DMOD. It is not called directly by the application.

The parameters for **sbpurcvx** should be taken from the parameters for routproc. Note, however, that the first parameter to **sbpurcvx** is a pointer to a pointer to a buffer header (that is, a pointer to the corresponding parameter for the routing procedure, not the parameter itself).

sbpusend

The **sbpusend** function sends a message from an application to a partner on an LPI connection.

Syntax

```
VOID sbpusend(
PTRBFHDR msgptr
);
```

Parameters

msgptr

Pointer to the message to be sent.

Remarks

The message buffer is released after transmission by the Dynamic Access Module (DMOD). It cannot be accessed by the application again.

For an Open request message, the **destl** parameter can be zero. In this case, the Resource Locator will attempt to find a suitable destination for the Open message.

sbputerm

The **sbputerm** function must be called when the application terminates. It frees the DL-BASE and Dynamic Access Module (DMOD) resources used by the application.

For Win32®, do not call **sbputerm** from an entry point in a detached DLL process because it may cause a deadlock inside the SNADMOD.DLL.

Syntax

```
VOID sbputerm(
void
);
```

sepdbubl

The application calls the **sepdbubl** function to get a buffer with a requested number of elements.

Syntax

```
PTRBFHDR sepdbubl(
USHORT noelts
);
```

Parameters

noelts

Number of elements required.

Return Value

A pointer to the buffer obtained. NULL if a buffer could not be obtained.

Remarks

Each element has a size of 268—the constant SNANBEDA in the header file FMI.H.

The returned buffer consists of a header and the required number of elements. The header points to the first element, which points to the next element, and so on to make an element chain.

It is possible to add an element to an existing buffer by calling sbpibegt to get the extra element. The new element should be added to the element chain of the buffer, and the "number of elements" count should be updated.

The application must release any buffers that are not transmitted.

sepdburl

The application calls the **sepdburl** function to release a buffer.

Syntax

```
VOID sepdburl(
PTRBFHDR msgptr
);
```

Parameters

msgptr

Pointer to the buffer to be released.

Remarks

It is important that buffers are released after use. This is done automatically when a message is transmitted. For messages received, it is the responsibility of the application to either release or reuse the buffer.

This function releases both the buffer header and any associated buffer elements. It is possible to release single elements from a buffer by using the function sbpiberl.

sepdchnk

The **sepdchnk** function gets the function management interface (FMI) chunk size. The application calls this function to obtain the chunk size that should be used on the FMI. For more information on FMI chunking, see Pacing and Chunking.

Syntax

```
VOID sepdchnk(
USHORT *pipesizeptr,
USHORT *chunksizeptr
);
```

Parameters

pipesizeptr

Size in bytes of the pipe between the application and the local node.

chunksizeptr

Dynamic Access Module (DMOD) chunk size in bytes.

Remarks

The application does not need to use the pipe size returned by this call. (It is included on this call because the local node uses the same call to obtain both the pipe size and the chunk size.)

sepdcrec

The **sepdcrec** function gets configuration information. The application calls this function to obtain the 3270 configuration information for the name with which the user logged on to the network operating system. The call also registers this user name in the service table.

Syntax

```
USHORT sepdcrec(
UCHAR *pBuffer,
USHORT length,
USHORT *numbytes
);
```

Parameters

pBuffer

Pointer to a buffer supplied by the application, in which configuration information is returned.

length

Size of the supplied buffer.

numbytes

Used by Host Integration Server 2009 to return the number of bytes of information returned in the buffer.

Return Value

NO_ERROR (0)

OK.

NOCSSRVR (1)

No configuration file server available.

NODGNREC (2)

No diagnostics record found in configuration file.

NOUSRREC (3)

No user record found in configuration file for this user.

BUF2SMAL (4)

Supplied buffer was too small.

NONOS (5)

Network operating system is not started.

NOTLOGON (6)

User is not logged on to the network operating system.

READERR (7)

Failed to read from configuration file.

NONAP (8)

The Network Access Program (NAP) is not started.

MAXAPP (9)

Windows only: Maximum number of concurrent applications exceeded.

ERROR_SERVER (14)

Error on the server end of the remote procedure call (RPC).

ERROR_LOCAL_FAILURE (15)

Error on the local end of the RPC.

Remarks

The sbpuinit function should always be called before any other DL-BASE or Dynamic Access Module (DMOD) entry points except SNAGetVersion. For new emulators, **sepdcrec** should be called after **sbpuinit**. (Because of the order of calls used in older emulators, a call to **sepdcrec** before **sbpuinit** is still supported, but this order is not recommended.)

On successful return, the buffer contains pointers to the appropriate 3270 user record and the diagnostics record, followed by the records themselves. It is formatted as follows:

```
TECWRKUS *pUserRecord,
TEDIAGNS *pDiagRecord
);
```

(UserRecord—variable length)

(DiagRecord)

The two records should be accessed using the supplied pointers.

See Configuration Information for details of the format of these records and of how the application uses the configuration file information.

If there is no 3270 user record for this user in the configuration file, or if no diagnostics record is found in the configuration file (an internal error), the application should terminate and not allow the user to use 3270 emulation. The Host Integration Server error log messages COM0438 and COM0437 can be used to report these failures.

If the supplied buffer is too small for the returned information, the contents of the buffer are undefined and should not be examined, but the *numbytes* parameter will contain the total number of bytes of information available (that is, the size of the two pointers plus the two configuration records). The application should retry with a buffer of at least this size.

sepdgetinfo

The **sepdgetinfo** function returns a structure containing the version number of Host Integration Server 2009, the path of the current configuration file, and the network operating system over which Host Integration Server is running.

Syntax

```
USHORT sepdgetinfo(
struct cs_info *pCSInfo
);
```

Parameters

pCSInfo

Pointer to a buffer supplied by the application, containing a **cs_info** data structure in which system information is returned. The application must set the **length** member in this data structure (for more information, see Remarks later in this topic); the other members should be set to nulls or blanks.

The cs_info structure

The returned **cs_info** structure and its members are as follows:

```
struct cs_info {
    unsigned short length;
    unsigned char  major_ver;
    unsigned char  minor_ver;
    unsigned char  config_share[80];
    unsigned short nos;
 } cs_info;
```

Members

length

Length of the data structure supplied by the application.

major_ver

Major version number:

1 for Host Integration Server 1.1 (Connection Server 1.1) 2 for Host Integration Server 2.0 (Connection Server 2.0)

minor_ver

Minor version number (decimal):

10 for Connection Server 1.1 (indicates 1.10)00 for Connection Server 2.0 (indicates 2.00)

config_share[80]

Path of the running configuration file：\\server\share\ (null terminated).

nos

Network operating system in use

1: LAN Manager / LAN Server2: NetWare

Return Value

NO_ERROR (0)

OK.

NOCSSRVR (1)

No configuration file server available.

BADLNGTH (2)

Supplied buffer was too small.

Remarks

The application must set the **length** member to the length of the **cs_info** structure (86 bytes in the current version). Any other value will be rejected. This parameter is used to ensure compatibility with future versions; an application supplying this length will always obtain the information shown here, but in future versions it may be possible to specify larger values and obtain further information.

On successful return, the data structure **cs_info** contains the version number of Host Integration Server 2009, the path of the current configuration file, and the network operating system over which Host Integration Server is running.

Do not use the configuration file path returned by **sepdgetinfo** because NetWare clients will not be able to access this path.

If there is no configuration file server available, only the version number fields are valid; the other fields should not be checked.

sepdrout

The **sepdrout** function for Win32® allows an application to perform its own routing of received messages by setting up a procedure that is called by the Dynamic Access Module (DMOD) when a message is received.

Syntax

```
DWORD sepdrout(
DWORD ( *proc_addr,)
(BUFHDR *, USHORT, USHORT
);

DWORD sepdrout(
    DWORD *proc_addr,
    (BUFHDR *, USHORT, USHORT
   );
```

Parameters

proc_addr

The routing procedure.

Return Value

NO_ERROR (0)

Successful.

Anything else

Unsuccessful.

Remarks

This facility is only available to clients, as defined in the call to sbpuinit.

An application can have up to four routing procedures. Note that the Advanced Program-to-Program Communications (APPC) and Common Service Verb (CSV) libraries each use a routing procedure. When the DMOD receives a message, each routing procedure is called, until one accepts the message.

For an example of a routing procedure, see routproc.

sepwrout

The **sepwrout** function is the Windows version of sepdrout. It has the same parameters and is used in exactly the same way.

FMI Message Formats

This section describes the message formats for the function management interface (FMI). The message formats are presented in a language-independent notation. Details of the message format notation and key assumptions about the contents of the message formats are as follows:

- Reserved indicates that the field is set to zero (for a numeric field) or all nulls (for names) by the sender of the message.
- Undefined indicates that the value of the field is indeterminate. The field is not set by the sender and should not be examined by the receiver of the message.
- Fields that occupy two bytes, such as **opresid** in the Open(PLU) Request, are represented with the most arithmetically significant byte in the lowest byte address, irrespective of the normal orientation used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. However, the following fields are exceptions:
    - The **srci** and **desti** fields in buffer headers are stored in the local format of the application that assigns them, because only the assigning application needs to interpret these values.
    - The **startd** and **endd** fields in elements are always stored in low-byte, high-byte orientation (the normal orientation of an Intel processor).
- Messages are composed of buffers consisting of a buffer header and zero or more buffer elements. For more information about buffer formats, see Messages.
- Applications must assign unique index (I) values for every active LPI connection within the node. In particular, the Open(SSCP) Request must be different from the source index it sends in response to the Open(PLU). Additionally, zero should not be used as an I value. An I value of zero means that the sender of the message is inviting the recipient of the message to assign an I value.
- The **startd** field in each element gives the offset of the first byte of data in the element after the **trpad** field.

  For non-logical unit application (LUA) applications, **startd** will either be 1 (data starts in the byte after the **trpad** field), 10 (nine bytes of padding are included between the **trpad** field and the start of the data), or 13 (12 bytes of padding are included between the **trpad** field and the start of the data).

  For LUA applications, **startd** is 4 (three bytes of padding between the **trpad** field and the start of the data) in the first element of a message and 13 (12 bytes of padding) in subsequent elements.

  The local node uses extra bytes for additional header information. This avoids having to copy data into a new buffer when adding this information.
- Because **startd** indicates the index into **dataru** starting from 1, not 0, the first byte of valid data is always at **dataru[startd–1]**.
- If **startd** is greater than **endd**, there is no valid data in the message.
- All fields within **dataru** are of type **CHAR**, except where the remarks indicate otherwise.
- Note that where a buffer element has a **startd** of 1, 10, or 13, this only applies to the initial element in the chain of elements, and subsequent elements in the chain have a **startd** of 1. Messages with two distinct linked element chains in the message formats (for example Open(PLU) Request and Open(PLU) OK Response) have the **startd** field in the elements at the start of the chains as the value (1, 10, or 13) given in the message format, and the **startd** fields in all other elements as 1.

In This Section

- Open(SSCP)
- Open(SSCP) Request
- Open(SSCP) Response
- Open(PLU)
- Open(PLU) Request
- Open(PLU) OK Response
- Open(PLU) Error Response
- Open(PLU) OK Confirm
- Open(PLU) Error Confirm
- Close(SSCP)
- Close(SSCP) Request
- Close(SSCP) Response
- Close(PLU)
- Close(PLU) Request
- Close(PLU) Response
- Data
- Status-Acknowledge
- Status-Acknowledge(Ack)
- Status-Acknowledge(Nack-1)
- Status-Acknowledge(Nack-2)
- Status-Acknowledge(ACKLUA)
- Status-Control
- Status-Control(...) Request
- Status-Control(...) Acknowledge
- Status-Control(...) Negative-Acknowledge-1
- Status-Control(...) Negative-Acknowledge-2
- Status-Control(...) ACKLUA
- Status-Error

- Status-Resource
- Status-RTM
- Status-Session

Open(SSCP)

The **Open(SSCP)** message is used by the application to open the system services control point (SSCP) connection. The Open request is sent by the application to the node, and the Open response comes from the node to the application.

Open(SSCP) Request

The **Open(SSCP) Request** message flows from the application to the node. It is used with a system services control point (SSCP) connection.

Syntax

```
  struct Open(SSCP) Request {
      PTRBFHDR  nxtqptr;
      PTRBFELT  hdrept;
      CHAR      numelts;
      CHAR      msgtype;
      CHAR      srcl;
      CHAR      srcp;
      INTEGER   srci;
      CHAR      destl;
      CHAR      destp;
      INTEGER   desti;
      CHAR      ophdr.openqual;
      CHAR      ophdr.opentype;
      CHAR      ophdr.appltype;
      CHAR      ophdr.opluno;
      INTEGER   ophdr.opresid;
      INTEGER   ophdr.icreditr;
      INTEGER   ophdr.icredits;
      CHAR      ophdr.opninfo1;
      CHAR      ophdr.opnpad1;
  };
  struct Open(SSCP) Request {
      PTRBFELT   hdreptr->elteptr;
      INTEGER    hdreptr->startd;
      INTEGER    hdreptr->endd;
      CHAR       hdreptr->trpad;
      CHAR[268]  hdreptr->dataru;
  };
  struct Open(SSCP) Request {
      PTRBFELT  hdreptr->elteptr->elteptr;
      INTEGER   hdreptr->elteptr->startd;
      INTEGER   hdreptr->elteptr->endd;
      CHAR      hdreptr->elteptr->trpad;
      CHAR[268] hdreptr->elteptr->dataru;
  };
```

Members

nxtqptr

Pointer to next buffer header.

hdrept

Pointer to first buffer element.

numelts

Number of buffer elements (0x02).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner. (For more information, see Remarks.)

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier REQU (0x01).

ophdr.opentype

Open type SSCPSEC (0x01).

ophdr.appltype

Application program interface type.

Function management interface (FMI) without chunking (0x02).

FMI with chunking (0x82). (For more information, see Remarks.)

ophdr.opluno

Logical Unit number. (For more information, see Remarks.)

ophdr.opresid

Resource identifier.

ophdr.icreditr

Reserved.

ophdr.icredits

Reserved.

ophdr.opninfo1

Reserved.

ophdr.opnpad1

Open force type. (For more information, see Remarks.)

OPEN_TEST (0x00)

OPEN_FORCE (0x01)

Element 1

hdreptr−>elteptr

Pointer to next buffer element.

hdreptr−>startd

Start of data in this buffer element (1).

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved (1 byte).

hdreptr−>dataru

Data request/response unit (RU), as follows:

dataru[0–9]

Source name. Should be filled with blanks.

dataru[10–19]

Destination name. Set to the logical unit (LU) with which you want to communicate.

dataru[20]

Sense 4003 flag.

dataru[21]

Sense 4004 flag.

dataru[22]

Sense 4006 flag.

dataru[23]

Sense 4007 flag.

dataru[24]

Sense 4009 flag.

dataru[25]

Sense 400A flag.

dataru[26]

Sense 400B flag.

dataru[27]

Sense 400C flag.

dataru[28]

Sense 400D flag.

dataru[29]

Sense 400F flag.

dataru[30]

Sense 4011 flag.

dataru[31]

Sense 4012 flag.

dataru[32]

Sense 4014 flag.

dataru[33]

High priority indicator.

HIGH (0x01)

LOW (0x02)

dataru[34]

Logical unit application (LUA) supported indicator.

Supported (0x01)

Not supported (0x00)

dataru[35–36]

Chunk size obtained from Dynamic Access Module (DMOD). (For more information, see Remarks.)

dataru[37]

Segment delivery option.

Do not deliver request/response unit (RU) segments (0x00)

Deliver RU segments (0x01)

dataru[38]

High-level language application programming interface (HLLAPI) session identifier. (For more information, see Remarks.)

Element 2

hdreptr–>elteptr–>elteptr

Pointer to next buffer element (NIL).

hdreptr–>elteptr–>startd

Start of data in this buffer element (1).

hdreptr–>elteptr–>endd

End of data in this buffer element.

hdreptr–>elteptr–>trpad

Reserved.

hdreptr–>elteptr–>dataru

Data RU, as follows:

dataru[0]

ASCII string identifying the 3270 emulator. (For more information, see Remarks.)

Remarks

- The **Open(SSCP) Request** message consists of a buffer header and two buffer elements.
- The source L value, the destination Locality Partner Index (LPI) values, and the source name are reserved.
- For a 3270 emulator, the source P value must be set to S3PROD (0x12), which identifies the application as a 3270 emulator. The destination name should be set to the LU name or pool name taken from the 3270 user record (right-filled with ASCII spaces if fewer than 10 characters).
- An LUA application uses the source P value LUAPROD (0x1D). This is independent of the value of the LUA supported indicator, which selects the LUA variant of the FMI.
- The SNS4003 to SNS4014 fields, together with the high priority indicator, are referred to in the text as the SSCP connection information control block (CICB). (For more information, see Opening the SSCP Connection.) A value of 0x00 indicates that the data flow control (DFC) receive check corresponding to the sense code is not supported for this LU. A value of 0x01 indicates that it is supported. Note that the corresponding send checks are always performed regardless of these values.
- The LU number is only used internally in the local node on the **Open(SSCP) Request**. It is generated from the destination name in the first element.
- The open force type field is used when locating resources across more than one server and for automatic activation of connections when the application wishes to use an LU for which the connection is inactive. The application does not need to set this flag. It is used by the DL-BASE. For details, see Opening the SSCP Connection.
- The application program interface type field defines whether RU chunking is used from the local node to the application. This may be necessary if large RUs are being used. For more information about FMI chunking, see Pacing and Chunking.
- The chunk size field (at **dataru[35]**) is an integer value.

- The segment delivery option specifies whether the local node should deliver segments of RUs to the application as soon as they are received or should assemble whole RUs before delivering them to the application. Segment delivery allows the application to update the user's screen as soon as data is received, known as window shading, which can result in a faster perceived response. For more information, see Segment Delivery. This option is required only when chunking is being used. It is included on this message so that the local node can calculate the initial chunk credit values on the corresponding primary logical unit (PLU) connection. The option must still be set on the **Open(PLU) Response**. The setting specified on that message will override the one specified here if there is a conflict. If this happens, the initial credit values may not be suitable.

- The LUA supported indicator specifies whether the application uses the LUA variant of the FMI.

- If the element is shorter than (s+34) bytes, Microsoft® Host Integration Server assumes no LUA and no chunking. This ensures backward compatibility with previous versions of the local node software in which these options were not available.

- The HLLAPI session identifier is a single ASCII character that identifies the 3270 display session to which the **Open(SSCP)** applies. HLLAPI uses this to identify a particular 3270 presentation space to which an HLLAPI function refers. It is also referred to by 3270 as the session's short name, or by HLLAPI as the presentation space identifier (PS identifier). If the 3270 emulator does not support session identifiers, this field should be set to zero.

- The second element contains an ASCII string that you can use to identify the type of 3270 emulator. This string will be logged in the audit log file by the client's DL-BASE and can also be seen in traces. The **startd** and **endd** fields must be set up to define the limits of this string.

Open(SSCP) Response

The **Open(SSCP) Response** message flows from the node to the application. It is used with an system services control point (SSCP) connection.

Syntax

```
  struct Open(SSCP) Response {
      PTRBFHDR  nxtqptr;
      PTRBFELT hdreptr;
      CHAR     numelts;
      CHAR     msgtype;
      CHAR     srcl;
      CHAR     srcp;
      INTEGER  srci;
      CHAR     destl;
      CHAR     destp;
      INTEGER  desti;
      CHAR     ophdr.openqual;
      CHAR     ophdr.opentype;
      CHAR     ophdr.appltype;
      CHAR     ophdr.opluno;
      INTEGER  ophdr.opresid;
      INTEGER  ophdr.operr1;
      INTEGER  ophdr.operr2;
  };
  struct Open(SSCP) Response {
      PTRBFELT  hdreptr->elteptr;
      INTEGER   hdreptr->startd;
      INTEGER   hdreptr->endd;
      CHAR      hdreptr->trpad;
      CHAR[256] dataru;
  };
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element.

numelts

Number of buffer elements (0x01).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier.

RSPOK (0x02) RSPERR (0x03)

ophdr.opentype

Open type SSCPSEC (0x01).

ophdr.appltype

Application program interface type.

0x02 (function management interface (FMI) application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.operr1

Error code 1.

ophdr.operr2

Error code 2.

Element 1

hdreptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>startd

Start of data in this buffer element (1).

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved (1 byte).

hdreptr–>dataru

Data RU, as follows:

dataru[0–9]

Source name.

dataru[10–19]

Destination name.

dataru[20–27]

Name of the local node that accepted the Open.

dataru[28–35]

Name of the connection used by the logical unit (LU).

dataru[36–37]

The internal identifier of the local node for the connection. (For more information, see Remarks.)

dataru[38]

The type of link service used by the connection, as shown in the following table.

Link service	Connection
CESLINK	(03) - SDLC
CESX25	(04) - X.25
CESTR	(11) - Token Ring
CESTCPIP	(30) - TCP/IP
CESRELAY	(31) - Frame Relay
CESCHANL	(32) - Channel
CESISDN	(33) – ISDN
CESETHER	(34) - Ethernet 802.2

Remarks

- The **Open(SSCP) Response** message consists of a buffer header and a single buffer element.
- If the open qualifier is RSPERR, the error code is valid and the Locality Partner Index (LPIs) and names are undefined. (For more information, see Error and Sense Codes.)
- The LU number indicates the LU selected by the local node from the configuration data. (For more information, see Opening the SSCP Connection.)
- When the **Open(SSCP)** is for an LU group, the source name contains the name of the selected LU.
- The connection identifier is an integer value. It uniquely identifies a particular connection on this local node. All sessions using the same connection will return the same identifier. This value is typically used when a link error is received on one session to determine which other sessions will be affected.

Open(PLU)

The **Open(PLU)** message is used by the local node to open the primary logical unit (PLU) connection with the application on receipt of a **BIND** command from the host.

Open(PLU) Request

The **Open(PLU) Request** message flows from the node to the application. It is used with a primary logical unit (PLU) connection.

```
struct Open(PLU) Request {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      ophdr.openqual;
    CHAR      ophdr.opentype;
    CHAR      ophdr.appltype;
    CHAR      ophdr.opluno;
    INTEGER   ophdr.opresid;
    INTEGER   ophdr.icreditr;
    INTEGER   ophdr.icredits;
    CHAR      ophdr.opninfo1;
};
struct Open(PLU) Request {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
struct Open(PLU) Request {
    PTRBFELT  hdreptr->elteptr->elteptr;
    INTEGER   hdreptr->elteptr->startd;
    INTEGER   hdreptr->elteptr->endd;
    CHAR      hdreptr->elteptr->trpad;
    CHAR[ ]   hdreptr->elteptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element.

numelts

Number of buffer elements (0x02).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier REQU (0x01).

ophdr.opentype

Open type LUSEC (0x02).

ophdr.appltype

Application program interface type.

0x02 (FMI application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.icreditr

Initial credit for flow from application to local node: zero (no flow control).

ophdr.icredits

Recommended initial credit for flow from local node to application: Pacing window + 1.

ophdr.opninfo1

Negotiable bind indicator.

Bind is not negotiable (0x00)

Bind is negotiable (0x01)

Element 1

hdreptr–>elteptr

Pointer to buffer element.

hdreptr–>startd

Start of data in this buffer element (1).

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU, as follows:

dataru[0–9]

Source name.

dataru[10–19]

Destination name.

dataru[20]

Secondary pacing send window.

dataru[21]

Secondary pacing receive window.

dataru[22–23]

Secondary send maximum request/response unit (RU) size. (For more information, see Remarks.)

dataru[24–25]

Primary send maximum RU size. (For more information, see Remarks.)

dataru[26]

Secondary send chunk size (in units of elements).

dataru[27]

Primary send chunk size (in units of elements).

Element 2

hdreptr–>elteptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>elteptr–>startd

Start of data in this buffer element (13).

hdreptr–>elteptr–>endd

End of data in this buffer element.

hdreptr–>elteptr–>trpad

Reserved.

hdreptr–>elteptr–>dataru

Data RU, as follows:

dataru[13]

The **BIND** RU received from the host.

Remarks

- The **Open(PLU) Request** message consists of a buffer header, an initial element containing the source and destination names, RU sizes, and so on, followed by a second element containing the **BIND** RU received from the host.
- The source Locality Partner Index (LPI) and the L and P parts of the destination LPI are valid, but the I part of the destination LPI is reserved.
- The two send maximum RU size fields (in **dataru[22–25]**) are both integer values.
- The **BIND** RU can be up to 256 bytes in length.
- If the application is using the logical unit application (LUA) variant of the function management interface (FMI), the **BIND** RU is preceded by its transmission header (TH) and response header (RH). The **startd** field of the second element points to the TH. (For more information about FMI, see FMI Concepts.)
- The LU number matches that allocated to the named application on the Open(SSCP) Response.
- The resource identifier matches the value used by the application on the Open(SSCP) Request.
- If chunking was specified on the **Open(SSCP) Request**, the **icredits** (initial credit from local node to application) field

specifies the number of chunks, rather than RUs, that can be transmitted. The two send chunk size parameters are specified in units of elements. (Each element contains up to 256 bytes of RU data.) A value of zero indicates that the chunk size is not the limiting factor in determining the size of messages. The limiting factor is the RU size or the segment size, so chunking is not required. In this case, credit will still be used, with the unit of credit being a message.

- The **icreditr** (initial credit from application to local node) field is not used and must be set to zero.

Open(PLU) OResponse

The **Open(PLU) OK Response** message flows from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Open(PLU) OK Response {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      ophdr.openqual;
    CHAR      ophdr.opentype;
    CHAR      ophdr.appltype;
    CHAR      ophdr.opluno;
    INTEGER   ophdr.opresid;
    INTEGER   ophdr.icreditr;
    INTEGER   ophdr.icredits;
    CHAR      ophdr.opninfo1;
};
struct Open(PLU) OK Response {
    PTRBFELT hdreptr->elteptr;
    INTEGER hdreptr->startd;
    INTEGER hdreptr->endd;
    CHAR hdreptr->trpad;
    CHAR[268] hdreptr->dataru;
};
struct Open(PLU) OK Response {
    PTRBFELT   hdreptr->elteptr->elteptr;
    INTEGER    hdreptr->elteptr->startd;
    INTEGER    hdreptr->elteptr->endd;
    CHAR       hdreptr->elteptr->trpad;
    CHAR[268]  hdreptr->elteptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element.

numelts

Number of buffer elements (0x02).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier RSPOK (0x02).

ophdr.opentype

Open type LUSEC (0x02).

ophdr.appltype

Application program interface type.

0x02 (FMI application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.icreditr

Initial credit for flow from application to local node: zero.

ophdr.icredits

Initial credit for flow from local node to application; only valid if APPLPAC = 0x01.

ophdr.opninfo1

Negotiable bind indicator.

Bind is not negotiable (0x00)

Bind is negotiable (0x01)

Element 1

hdreptr–>elteptr

Pointer to buffer element.

hdreptr–>startd

Start of data in this buffer element (1).

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU, as follows:

dataru[0–9]

Source name.

dataru[10–19]

Destination name.

dataru[20]

Segment delivery option.

Do not deliver request/response unit (RU) segments (0x00)

Deliver RU segments (0x01)

dataru[21]

Application pacing option.

No application pacing (0x00)

Application pacing (0x01)

dataru[22]

Application cancel option: Cancel is generated by:

local node (0x00)

application (0x01)

dataru[23]

Application transaction numbers option: transaction numbers are:

not supported by application (0x00)

supported by application (0x01)

dataru[24]

BIND table index

BIND_TABLE_INDEX_PRT (1) (printer session)

BIND_TABLE_INDEX_CRT (2) (display session)

Element 2

hdreptr–>elteptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>elteptr–>startd

Start of data in this buffer element (13).

hdreptr–>elteptr–>endd

End of data in this buffer element.

hdreptr–>elteptr–>trpad

Reserved.

hdreptr–>elteptr–>dataru

Data RU, as follows:

dataru[13]

The **BIND** RU.

Remarks

- The **Open(PLU) OK Response** message consists of a buffer header, an initial element containing the source and destination names and connection information control block (CICB), followed by elements containing the **BIND** RU received from the host.

- The application should reflect the source and destination Locality Partner Index (LPIs) and the source and destination names from the Open(PLU) Request and must supply the I part of the source LPI.

- The fields from segment delivery option to bind table index (in the first element) are referred to in the text as the PLU CICB. For more information about the contents of the CICB, see Opening the PLU Connection.

- The **BIND** RU can be up to 256 bytes in length.

- For LUA, the **BIND** RU is not preceded by its transmission header (TH) and response header (RH). This is in contrast with the Open(PLU) Request, where the TH and RH are included.

- As in the **Open(PLU) Request**, the **icredits** value is in units of chunks if chunking is being used.

Open(PLU) Error Response

The **Open(PLU) Error Response** message flows from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Open(PLU) Error Response {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      ophdr.openqual;
    CHAR      ophdr.opentype;
    CHAR      ophdr.appltype;
    CHAR      ophdr.opluno;
    INTEGER  ophdr.opresid;
    INTEGER  ophdr.operr1;
    INTEGER  ophdr.operr2;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier RSPERR (0x03).

ophdr.opentype

Open type LUSEC (0x02).

ophdr.appltype

Application program interface type.

0x02 (FMI application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.operr1

Error code 1.

ophdr.operr2

Error code 2.

Remarks

- The **Open(PLU) Error Response** message consists of a buffer header only.
- The application should reflect the source and destination Locality Partner Index (LPIs).

Open(PLU) OConfirm

The **Open(PLU) OK Confirm** message flows from the node to the application. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Open(PLU) OK Confirm {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   dsti;
    CHAR      ophdr.openqual;
    CHAR      ophdr.opentype;
    CHAR      ophdr.appltype;
    CHAR      ophdr.opluno;
    INTEGER   ophdr.opresid;
    INTEGER   ophdr.icreditr;
    INTEGER   ophdr.icredits;
    CHAR      ophdr.opninfo1;
};
struct Open(PLU) OK Confirm {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element.

numelts

Number of buffer elements (0x01).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

dsti

Destination index.

ophdr.openqual

Open qualifier CONFOK (0x04).

ophdr.opentype

Open type LUSEC (0x02).

ophdr.appltype

Application program interface type.

0x02 (FMI application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.icreditr

Reserved.

ophdr.icredits

Reserved.

ophdr.opninfo1

PLU address.

Element

hdreptr−>elteptr

Pointer to buffer element (NIL).

hdreptr−>startd

Start of data in this buffer element (1).

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved.

hdreptr−>dataru

Data request/response unit (RU), as follows:

dataru[0]

Function management (FM) profile.

dataru[1]

Transmission Service profile (TS profile).

dataru[2]

Primary chaining use.

dataru[3]

Primary request control mode.

dataru[4]

Primary chain response protocol.

dataru[5]

Primary two-phase commit.

dataru[6]

Primary compression indicator.

dataru[7]

Primary send end bracket (EB) indicator.

dataru[8]

Secondary chaining use.

dataru[9]

Secondary request control mode.

dataru[10]

Secondary chain response protocol.

dataru[11]

Secondary two-phase commit.

dataru[12]

Secondary compression indicator.

dataru[13]

Secondary send EB indicator.

dataru[14]

Function Management Header (FMH) usage.

dataru[15]

Bracket usage.

Brackets not used (0x00)

Brackets used (0x01)

dataru[16]

Bracket reset state.

Bracket reset state between-brackets (BETB) (0x01)

Bracket reset state in-bracket (INB) (0x02)

dataru[17]

Bracket termination rule.

dataru[18]

Alternate code set indicator.

dataru[19]

Sequence number availability.

dataru[20]

Normal-flow send/receive mode.

dataru[21]

Half-duplex flip-flop reset.

dataru[22]

Secondary pacing send window.

dataru[23]

Secondary pacing receive window.

dataru[24–25]

Secondary send maximum RU size (INTEGER value).

dataru[26–27]

Primary send maximum RU size (INTEGER value).

dataru[28]

LU-LU session type.

dataru[29]

PLU name size.

dataru[30–37]

PLU name (Extended Binary Coded Decimal Interchange Code or EBCDIC).

dataru[38]

Session type 1: PS Function Management Header (FMH) type.

dataru[39]

PS data stream profile.

dataru[40]

Number of outstanding destinations.

dataru[41]

Compacted data indicator.

dataru[42]

Peripheral Data Interchange Record (PDIR) allowed indicator.

dataru[43]

Session type 2 or 3: query support.

dataru[44]

Dynamic screen size.

dataru[45]

Basic row size.

dataru[46]

Basic column size.

dataru[47]

Alternate row size.

dataru[48]

Alternate column size.

Remarks

- The **Open(PLU) OK Confirm** message consists of a buffer header and one element.
- The message does not carry source and destination names. Both LPIs are valid.
- The contents of **dataru** are referred to in the text as the PLU bind information control block (BICB). The BICB is only valid

for an open-qualifier of CONFOK. For further information about the contents of the BICB, see Opening the PLU Connection.

Open(PLU) Error Confirm

The **Open(PLU) Error Confirm** message flows from the node to the application. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Open(PLU) Error Confirm {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      ophdr.openqual;
    CHAR      ophdr.opentype;
    CHAR      ophdr.appltype;
    CHAR      ophdr.opluno;
    INTEGER   ophdr.opresid;
    INTEGER   ophdr.operr1;
    INTEGER   ophdr.operr2;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to first buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type OPENMSG (0x01).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

ophdr.openqual

Open qualifier CONFERR (0x05).

ophdr.opentype

Open type LUSEC (0x02).

ophdr.appltype

Application program interface type.

0x02 (function management interface (FMI) application)

ophdr.opluno

Logical unit number.

ophdr.opresid

Resource identifier.

ophdr.operr1

Error code 1.

ophdr.operr2

Error code 2.

Remarks

- The **Open(PLU) Error Confirm** message consists of a buffer header only.
- The error codes are valid. (For more information, see Error and Sense Codes.) An **Open(PLU) Error Confirm** closes the connection.

Close(SSCP)

The **Close(SSCP)** message closes an open system services control point (SSCP) connection.

Close(SSCP) Request

The **Close(SSCP) Request** message flows from the application to the node. It is used with a system services control point (SSCP) connection.

Syntax

```
struct Close(SSCP) Request {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      clhdr.closqual;
    CHAR      clhdr.clstype;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type CLOSEMSG (0x02).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

clhdr.closqual

Close qualifier REQU (0x01).

clhdr.clstype

Close subtype SSCPSEC (0x01).

Remarks

- The **Close(SSCP) Request** message consists of a buffer header only. There is no buffer element.

Close(SSCP) Response

The **Close(SSCP) Response** message flows from the node to the application. It is used with a system services control point (SSCP) connection.

Syntax

```
struct Close(SSCP) Response {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      clhdr.closqual;
    CHAR      clhdr.clstype;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type CLOSEMSG (0x02).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

clhdr.closqual

Close qualifier RSPOK (0x02).

clhdr.clstype

Close subtype SSCPSEC (0x01).

Remarks

- The **Close(SSCP) Response** message consists of a buffer header only. There is no buffer element.
- The **Close(SSCP)** protocol is unconditional. It is not possible for the local node to keep the SSCP connection open after the application sends **Close(SSCP)**.

Close(PLU)

The **Close(PLU)** message closes an open primary logical unit (PLU) connection.

Close(PLU) Request

The **Close(PLU) Request** message flows from the node to the application and from the application to the node. It is used with a primary logical unit (PLU) connection.

```
struct Close (PLU) Request {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      clhdr.closqual;
    CHAR      clhdr.clstype;
    CHAR      clhdr.clsctl;
    CHAR      clhdr.clspad1;
    INTEGER   clhdr.clspad2;
    INTEGER   clhdr.clserr1;
};
struct Close (PLU) Request {
    PTRBFELT hdreptr->elteptr;
    INTEGER hdreptr->startd;
    INTEGER hdreptr->endd;
    CHAR hdreptr->trpad;
    CHAR[268] hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type CLOSEMSG (0x02).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

clhdr.closqual

Close qualifier REQU (0x01).

clhdr.clstype

Close subtype LUSEC (0x02).

clhdr.clsctl

Close control

CLNORMAL	(0x01)	normal
CLBIND	(0x02)	bind forthcoming
CLCFAERR	(0x03)	CFA error
CLPUINAC	(0x04)	PU inactive
CLLUINAC	(0x05)	LU inactive
CLLNKERR	(0x06)	link error
CLBFSHRT	(0x07)	node buffer shortage
CLRCVCHK	(0x08)	DFC receive check
CLSLUTRM	(0x09)	SLU termination

clhdr.clspad1

Reserved.

clhdr.clspad2

Reserved.

clhdr.clserr1

Error code (only valid for close control = link error).

LUA only (see Remarks): Element

hdreptr−>elteptr

Pointer to buffer element (NIL).

hdreptr−>startd

Start of data in this buffer element (13).

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved.

hdreptr−>dataru

The **UNBIND** request/response unit (RU) received from the host, with its transmission header (TH) and response header (RH).

Remarks

- If the application is using the LUA variant of the function management interface (FMI), and the **Close(PLU) Request** was generated by receipt of an **UNBIND** from the host, the element is included, and **startd** points to the TH of the **UNBIND** message. (For more information about FMI, see FMI Concepts.)

- In all other cases (for example, if the **Close(PLU) Request** was generated by the local node as a result of a link outage), the message consists of a buffer header only. There is no buffer element.

- The close control field is only valid on messages from the local node to the application.

- If the close control field specifies link error, the error code field gives the link outage code.

Close(PLU) Response

The **Close(PLU) Response** message flows from the node to the application and from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Close(PLU) Response {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      clhdr.closqual;
    CHAR      clhdr.clstype;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type CLOSEMSG (0x02).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

clhdr.closqual

Close qualifier RSPOK (0x02).

clhdr.clstype

Close subtype LUSEC (0x02).

Remarks

- The **Close(PLU) Response** message consists of a buffer header only. There is no buffer element.

- The **Close(PLU)** protocol is unconditional. It is not possible for the recipient of a Close(PLU) Request (either the local node or an application) to keep the PLU connection open. The only valid response is **Close(PLU) Response** with the close qualifier as RSPOK.

Data

Data messages carry both inbound and outbound data between the application and the local node on all connections. For a detailed description of outbound and inbound data flows, see Data Flow.

The **Data** message flows from the node to the application and from the application to the node. It is used with both the system services control point (SSCP) and the primary logical unit (PLU) connections.

Syntax

```
struct Data {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      dfhdr.fhackrqd;
    CHAR      dfhdr.fhpad1;
    INTEGER   dfhdr.fhmsgkey;
    CHAR      dfhdr.fhflags1;
    CHAR      dfhdr.fhflags2;
    INTEGER   dfhdr.fhpad2;
    INTEGER   dfhdr.fhpad3;
    INTEGER   dfhdr.fhseqno;
};
struct Data {
    PTRBFELT   hdreptr->elteptr
    INTEGER    hdreptr->startd
    INTEGER    hdreptr->endd
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element.

numelts

Number of buffer elements.

msgtype

Message type DATAFMI (0x20).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

dfhdr.fhackrqd

Acknowledgment required indicator.

NOACKREQ (0x00) ACKREQ (0x01)

dfhdr.fhpad1

Reserved.

dfhdr.fhmsgkey

Message key.

dfhdr.fhflags1

Application flag 1.

dfhdr.fhflags2

Application flag 2.

dfhdr.fhpad2

Reserved.

dfhdr.fhpad3

Reserved.

dfhdr.fhseqno

Sequence number.

Element

hdreptr–>elteptr

Pointer to buffer element.

hdreptr–>startd

Start of data in this buffer element:

Non-logical unit application (LUA): 13, or 10 for second and subsequent segments of outbound segmented request/response units (RUs). LUA, inbound data: 4 in first element, 13 in subsequent elements.

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU.

Remarks

- The use of the acknowledgment required indicator in both inbound and outbound data acknowledgment protocols is described in Data Flow.
- The use of the application flag fields is described in Application Flags (For more information, see the note that follows for LUA.)
- The sequence number is undefined for inbound data but contains the corresponding SNA sequence number for

outbound data.

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH) and (if appropriate) response header (RH) are included in the data, and the **startd** field points to the TH. The **fhmsgkey**, **fhflags1**, **fhflags2**, and **fhseqno** fields are undefined and should not be used. The corresponding data from the element should be used instead. (For more information about FMI, see FMI Concepts.)

Status-Acknowledge

Status-Acknowledge messages flow between the application and the local node as part of the outbound and inbound data acknowledgment protocols. For a detailed description of outbound and inbound acknowledgment protocols, see Data Flow.

Note
The format of this message is slightly different for messages from the application to the local node on the primary logical unit (PLU) connection, as is explained in the following topic Status-Acknowledge(Ack)

Status-Acknowledge(Ack)

The **Status-Acknowledge(Ack)** message flows from the node to the application and from the application to the node, and is used with both system services control point (SSCP) and primary logical unit (PLU) connections.

The following structure shows the message format for all SSCP messages and for PLU messages flowing from the node to the application.

Syntax

```
struct Status-Acknowledge(Ack) {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stackhdr.akstat;
    CHAR      sfhdr.stackhdr.akqual;
    INTEGER   sfhdr.stackhdr.akmsgkey;
    CHAR      sfhdr.stackhdr.akflags1;
    CHAR      sfhdr.stackhdr.akflags2;
    INTEGER   sfhdr.stackhdr.aknumb1;
    INTEGER   sfhdr.stackhdr.aknumb2;
    INTEGER   sfhdr.stackhdr.akseqno;
};
struct Status-Acknowledge(Ack) {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stackhdr.akstat

Status type ACK (0x01).

sfhdr.stackhdr.akqual

Acknowledgment type ACKPOS (0x02).

sfhdr.stackhdr.akmsgkey

Message key.

sfhdr.stackhdr.akflags1

Application flag 1.

sfhdr.stackhdr.akflags2

Application flag 2.

sfhdr.stackhdr.aknumb1

Undefined.

sfhdr.stackhdr.aknumb2

Reserved.

sfhdr.stackhdr.akseqno

SNA sequence number.

LUA only (see Remarks): Element

hdreptr−>elteptr

Pointer to buffer element (NIL).

hdreptr−>startd

Start of data in this buffer element.

13 or 10 for second and subsequent segments of outbound segmented request/response units (RUs)

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved.

hdreptr−>dataru

Data RU.

The message format for PLU messages flowing from the application to the node is identical to the preceding format, except that the application flag 1 and application flag 2 fields are not used. They are replaced by the following INTEGER field:

sfhdr.stackhdr.akmsgtim

Last host response time

0xFFFF - no response time measured 0x*nnnn* - last response time measured, in units of 0.1 second

INTEGER	*sfhdr.stackhdr.akmsgtim*	Last host response time 0xFFFF - no response time measured 0x*nnnn* - last response time measured, in units of 0.1 second

Remarks

- The message key and application flags reflect the message key and application flags of the data message to which this is an acknowledgment. (For more information, see the note about LUA that follows.)

- For outbound **Status-Acknowledge(Ack)** messages from the local node to the application, the SNA sequence number gives the sequence number of the inbound data message to which this is an acknowledgment. (For more information, see the note about LUA that follows.) It is normally used only by Transmission Service profile (TS profile) 4 applications.

- For inbound **Status-Acknowledge(Ack)** messages from the application to the local node, the SNA sequence number reflects the sequence number of the outbound data message to which this is an acknowledgment.

- If the host specified that response time statistics are to be maintained, the application is responsible for measuring and reporting response times to the local node, using the **akmsgtim** field of this message. (For details, see RTM Parameters and Response Time Monitor Data.)

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH) and (if appropriate) response header (RH) are included in the data, and the **startd** field points to the TH. The **akmsgkey**, **akflags1**, and **akflags2** fields are undefined and should not be used. The corresponding data from the element should be used instead. The **akseqno** field is similarly undefined on messages from the local node to the application. It must be set on messages from the application to the local node. The **akseqno** field is used to hold the sequence number of the request being acknowledged. (For more information about FMI, see FMI Concepts.)

- If the application is not using the LUA variant of the FMI, the message consists of a buffer header only. There is no buffer element.

Status-Acknowledge(Nack-1)

The **Status-Acknowledge(Nack-1)** message flows from the node to the application and from the application to the node. It is used with both system services control point (SSCP) and primary logical unit (PLU) connections.

Syntax

```
struct Status-Acknowledge(Nack-1) {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stackhdr.akstat;
    CHAR      sfhdr.stackhdr.akqual;
    INTEGER   sfhdr.stackhdr.akmsgkey;
    CHAR      sfhdr.stackhdr.akflags1;
    CHAR      sfhdr.stackhdr.akflags2;
    INTEGER   sfhdr.stackhdr.aknumb1;
    INTEGER   sfhdr.stackhdr.aknumb2;
    INTEGER   sfhdr.stackhdr.akseqno;
};
struct Status-Acknowledge(Nack-1) {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stackhdr.akstat

Status type ACK (0x01).

sfhdr.stackhdr.akqual

Acknowledgment type ACKNEG1 (0x03).

sfhdr.stackhdr.akmsgkey

Message key.

sfhdr.stackhdr.akflags1

Application flag 1.

sfhdr.stackhdr.akflags2

Application flag 2.

sfhdr.stackhdr.aknumb1

Sense data 1.

sfhdr.stackhdr.aknumb2

Sense data 2.

sfhdr.stackhdr.akseqno

SNA sequence number.

LUA only (see Remarks): Element

hdreptr−>elteptr

Pointer to buffer element (NIL).

hdreptr−>startd

Start of data in this buffer element.

13 or 10 for second and subsequent segments of outbound segmented request/response units (RUs)

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved.

hdreptr−>dataru

Data RU.

Remarks

- The message key and application flags reflect the message key and application flags of the data message to which this is a negative acknowledgment. (For more information, see the note about LUA that follows.)
- For **Status-Acknowledge(Nack-1)** messages from the local node to the application, the sense data reflects the sense data in the SNA negative response.
- For **Status-Acknowledge(Nack-1)** messages from the application to the local node, the sense data fields are those intended for the SNA negative response to the host.

- For outbound **Status-Acknowledge(Nack-1)** messages from the local node to the application, the SNA sequence number gives the sequence number of the inbound data message to which this is a negative acknowledgment. (For more information, see the note about LUA that follows.)

- For inbound **Status-Acknowledge(Nack-1)** messages from the application to the local node, the SNA sequence number reflects the sequence number of the outbound data message to which this is a negative acknowledgment.

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH) and (if appropriate) response header (RH) are included in the data, and the **startd** field points to the TH. The **akmsgkey**, **akflags1**, and **akflags2** fields are undefined and should not be used. The corresponding data from the element should be used instead. The **akseqno** field is similarly undefined on messages from the local node to the application. It must be set on messages from the application to the local node. (For more information about FMI, see FMI Concepts.)

- If the application is not using the LUA variant of the FMI, the message consists of a buffer header only. There is no buffer element.

Status-Acknowledge(Nack-2)

The **Status-Acknowledge(Nack-2)** message flows from the node to the application. It is used with both system services control point (SSCP) and primary logical unit (PLU) connections.

Syntax

```
struct Status-Acknowledge(Nack-2) {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stackhdr.akstat;
    CHAR      sfhdr.stackhdr.akqual;
    INTEGER   sfhdr.stackhdr.akmsgkey;
    CHAR      sfhdr.stackhdr.akflags1;
    CHAR sfhdr.stackhdr.akflags2;
    INTEGER sfhdr.stackhdr.aknumb1;
    INTEGER sfhdr.stackhdr.aknumb2;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stackhdr.akstat

Status type ACK (0x01).

sfhdr.stackhdr.akqual

Acknowledgment type ACKNEG2 (0x04).

sfhdr.stackhdr.akmsgkey

Message key.

sfhdr.stackhdr.akflags1

Reserved.

sfhdr.stackhdr.akflags2

Critical failure indicator.

Noncritical failure (0x00) Critical failure (0x01)

sfhdr.stackhdr.aknumb1

Error code 1.

sfhdr.stackhdr.aknumb2

Error code 2.

Remarks

- The **Status-Acknowledge(Nack-2)** message consists of a buffer header only. There is no buffer element.
- The message key refers to the message key in the inbound data message to which this is a negative acknowledgment.
- For more information about error codes, see Error and Sense Codes.

Status-Acknowledge(ACKLUA)

The **Status-Acknowledge(ACKLUA)** message is for logical unit application (LUA) applications only. It flows from the node to the application, and is used with both the system services control point (SSCP) and primary logical unit (PLU) connections.

Syntax

```
struct Status-Acknowledge(ACKLUA) {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stackhdr.akstat;
    CHAR      sfhdr.stackhdr.akqual;
    INTEGER   sfhdr.stackhdr.akmsgkey;
    CHAR      sfhdr.stackhdr.akflags1;
    CHAR      sfhdr.stackhdr.akflags2;
    INTEGER   sfhdr.stackhdr.aknumb1;
    INTEGER   sfhdr.stackhdr.aknumb2;
    INTEGER   sfhdr.stackhdr.akseqno;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stackhdr.akstat

Status type ACK (0x01).

sfhdr.stackhdr.akqual

Acknowledgment type.

sfhdr.stackhdr.akmsgkey

Message key.

sfhdr.stackhdr.akflags1

Application flag 1.

sfhdr.stackhdr.akflags2

Application flag 2.

sfhdr.stackhdr.aknumb1

Number of replies.

sfhdr.stackhdr.aknumb2

Reserved.

sfhdr.stackhdr.akseqno

SNA sequence number.

Remarks

- The message key and application flags are undefined and should not be checked.
- The SNA sequence number gives the sequence number of the inbound data message to which this is an acknowledgment.

Status-Control

For details about **Status-Control** message usage and for a summary of control type codes, see Status-Control Message.

Status-Control(...) Request

The **Status-Control(...) Request** message flows from the node to the application and from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Status-Control(...) Request {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stctlhdr.ctlstat;
    CHAR      sfhdr.stctlhdr.ctlqual;
    CHAR      sfhdr.stctlhdr.ctltype;
    CHAR      sfhdr.stctlhdr.ctlack;
    CHAR      sfhdr.stctlhdr.ctlflag1;
    CHAR      sfhdr.stctlhdr.ctlflag2;
    INTEGER   sfhdr.stctlhdr.ctlnumb1;
    INTEGER   sfhdr.stctlhdr.ctlnumb2;
    INTEGER   sfhdr.stctlhdr.ctlmsgk;
};
struct Status-Control(...) Request {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stctlhdr.ctlstat

Status type STCNTRL (0x02).

sfhdr.stctlhdr.ctlqual

Control qualifier (0x01).

sfhdr.stctlhdr.ctltype

Control type.

sfhdr.stctlhdr.ctlack

Acknowledgment required indicator.

No acknowledgment required (0x00) Acknowledgment required (0x01)

sfhdr.stctlhdr.ctlflag1

Application flag 1.

sfhdr.stctlhdr.ctlflag2

Application flag 2. (For more information, see STSN.)

sfhdr.stctlhdr.ctlnumb1

Code 1.

sfhdr.stctlhdr.ctlnumb2

Code 2.

sfhdr.stctlhdr.ctlmsgk

Message key.

LUA only (see Remarks): Element

hdreptr−>elteptr

Pointer to buffer element (NIL).

hdreptr−>startd

Start of data in this buffer element.

13 or 10 for second and subsequent segments of outbound segmented request/response units (RUs)

hdreptr−>endd

End of data in this buffer element.

hdreptr−>trpad

Reserved.

hdreptr−>dataru

Data RU.

Remarks

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH), response header (RH), and RU are included in the data element, and the **startd** field points to the TH. The **ctlflag1** and **ctlflag2** bytes are not defined and should not be used. The appropriate values from the data should be used instead. (For more information about FMI, see FMI Concepts.)

- If the application is not using the LUA variant of the function management interface (FMI), the message consists of a

buffer header only. There is no buffer element.

- For a summary of **Status-Control** control type codes, see the table in Status-Control Message
- The code 1 and code 2 fields apply only for **Status-Control** LUSTAT, SIGNAL, and STSN messages.
- The application flag byte 2 is used for the **Status-Control** STSN control byte. (For more information, see Recovery.)

Status-Control(...) Acknowledge

The **Status-Control(...) Acknowledge** message flows from the node to the application and from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Status-Control(...) Acknowledge {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stctlhdr.ctlstat;
    CHAR      sfhdr.stctlhdr.ctlqual;
    CHAR      sfhdr.stctlhdr.ctltype;
    CHAR      sfhdr.stctlhdr.ctlack;
    CHAR      sfhdr.stctlhdr.ctlflag1;
    CHAR      sfhdr.stctlhdr.ctlflag2;
    INTEGER   sfhdr.stctlhdr.ctlnumb1;
    INTEGER   sfhdr.stctlhdr.ctlnumb2;
    INTEGER   sfhdr.stctlhdr.ctlmsgk;
};
struct Status-Control(...) Acknowledge {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stctlhdr.ctlstat

Status type STCNTRL (0x02).

sfhdr.stctlhdr.ctlqual

Control qualifier ACKPOS (0x02).

sfhdr.stctlhdr.ctltype

Control type.

sfhdr.stctlhdr.ctlack

Reserved.

sfhdr.stctlhdr.ctlflag1

Application flag 1.

sfhdr.stctlhdr.ctlflag2

Application flag 2. (For more information, see STSN.)

sfhdr.stctlhdr.ctlnumb1

Code 1.

sfhdr.stctlhdr.ctlnumb2

Code 2.

sfhdr.stctlhdr.ctlmsgk

Message key.

LUA only (see Remarks): Element

hdreptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>startd

Start of data in this buffer element.

13 or 10 for second and subsequent segments of outbound segmented request/response units (RUs)

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU.

Remarks

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH), response header (RH), and RU are included in the data element, and the **startd** field points to the TH. The **ctlflag1** and **ctlflag2** bytes are not defined and should not be used. The appropriate values from the data should be used instead. (For more information about FMI, see FMI Concepts.)

- If the application is not using the LUA variant of the FMI, the message consists of a buffer header only. There is no buffer element.

- For a summary of **Status-Control** control type codes, see the table in Status-Control Message.
- The code 1 and code 2 fields apply only for **Status-Control(STSN) Acknowledge** messages, where they are the application's secondary-to-primary and primary-to-secondary sequence numbers respectively.
- For messages from the application to the local node, the message key field must match the message key in the corresponding **Status-Control Request**.

Status-Control(...) Negative-Acknowledge-1

The **Status-Control(...) Negative-Acknowledge-1** message flows from the node to the application and from the application to the node. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Status-Control(...) Negative-Acknowledge-1 {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stctlhdr.ctlstat;
    CHAR      sfhdr.stctlhdr.ctlqual;
    CHAR      sfhdr.stctlhdr.ctltype
    CHAR      sfhdr.stctlhdr.ctlack;
    CHAR      sfhdr.stctlhdr.ctlflag1;
    CHAR      sfhdr.stctlhdr.ctlflag2;
    INTEGER   sfhdr.stctlhdr.ctlnumb1;
    INTEGER   sfhdr.stctlhdr.ctlnumb2;
    INTEGER   sfhdr.stctlhdr.ctlmsgk;
};
struct Status-Control(...) Negative-Acknowledge-1 {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL if not using LUA).

numelts

Number of buffer elements (0x00 if not using LUA).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stctlhdr.ctlstat

Status type STCNTRL (0x02).

sfhdr.stctlhdr.ctlqual

Control qualifier ACKNEG1 (0x03).

sfhdr.stctlhdr.ctltype

Control type.

sfhdr.stctlhdr.ctlack

Reserved.

sfhdr.stctlhdr.ctlflag1

Application flag 1.

sfhdr.stctlhdr.ctlflag2

Application flag 2.

sfhdr.stctlhdr.ctlnumb1

Sense code 1.

sfhdr.stctlhdr.ctlnumb2

Sense code 2.

sfhdr.stctlhdr.ctlmsgk

Message key.

LUA only (see Remarks): Element

hdreptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>startd

Start of data in this buffer element.

13 or 10 for second and subsequent segments of outbound segmented request/response units (RUs)

hdreptr–>endd

End of data in this buffer element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU.

Remarks

- If the application is using the LUA variant of the function management interface (FMI), the transmission header (TH), response header (RH), and RU are included in the data element, and the **startd** field points to the TH. The **ctlflag1** and **ctlflag2** bytes are not defined and should not be used. The appropriate values from the data should be used instead. (For more information about FMI, see FMI Concepts.)

- If the application is not using the LUA variant of the FMI, the message consists of a buffer header only. There is no buffer element.

- For messages from the application to the local node, the message key field must match the message key in the corresponding **Status-Control** request.

- For a summary of **Status-Control** control type codes, see the table in Status-Control Message.

Status-Control(...) Negative-Acknowledge-2

The **Status-Control(...) Negative-Acknowledge-2** message flows from the node to the application. It is used with a primary logical unit (PLU) connection.

Syntax

```
struct Status-Control(...) Negative-Acknowledge-2 {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stctlhdr.ctlstat;
    CHAR      sfhdr.stctlhdr.ctlqual;
    CHAR      sfhdr.stctlhdr.ctltype
    CHAR      sfhdr.stctlhdr.ctlack;
    CHAR      sfhdr.stctlhdr.ctlflag1;
    CHAR      sfhdr.stctlhdr.ctlflag2;
    INTEGER   sfhdr.stctlhdr.ctlnumb1;
    INTEGER    sfhdr.stctlhdr.ctlnumb2;
    INTEGER   sfhdr.stctlhdr.ctlmsgk;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stctlhdr.ctlstat

Status type STCNTRL (0x02).

sfhdr.stctlhdr.ctlqual

Control qualifier ACKNEG2 (0x04).

sfhdr.stctlhdr.ctltype

Control type.

sfhdr.stctlhdr.ctlack

Reserved.

sfhdr.stctlhdr.ctlflag1

Reserved.

sfhdr.stctlhdr.ctlflag2

Reserved.

sfhdr.stctlhdr.ctlnumb1

Error code 1.

sfhdr.stctlhdr.ctlnumb2

Error code 2.

sfhdr.stctlhdr.ctlmsgk

Message key.

Remarks

- The **Status-Control() Negative-Acknowledge-2** message consists of a buffer header only. There is no buffer element.
- The message key field matches the message key in the corresponding **Status-Control** request.
- For a summary of **Status-Control** control type codes, see the table in Status-Control Message.
- For a list of error codes, see Error and Sense Codes.

Status-Control(...) ACKLUA

The **Status-Control(...) ACKLUA** message is for logical unit application (LUA) applications only. It flows from the node to the application, and is used with a primary logical unit (PLU) connection.

Syntax

```
struct Status-Control(...) ACKLUA {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stctlhdr.ctlstat;
    CHAR      sfhdr.stctlhdr.ctlqual;
    CHAR      sfhdr.stctlhdr.ctltype
    CHAR      sfhdr.stctlhdr.ctlack;
    CHAR      sfhdr.stctlhdr.ctlflag1;
    CHAR      sfhdr.stctlhdr.ctlflag2;
    INTEGER   sfhdr.stctlhdr.ctlnumb1;
    INTEGER   sfhdr.stctlhdr.ctlnumb2;
    INTEGER   sfhdr.stctlhdr.ctlmsgk;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stctlhdr.ctlstat

Status type STCNTRL (0x02).

sfhdr.stctlhdr.ctlqual

Control qualifier ACKLUA (0x05).

sfhdr.stctlhdr.ctltype

Control type.

sfhdr.stctlhdr.ctlack

Reserved.

sfhdr.stctlhdr.ctlflag1

Application flag 1.

sfhdr.stctlhdr.ctlflag2

Application flag 2.

sfhdr.stctlhdr.ctlnumb1

Code 1.

sfhdr.stctlhdr.ctlnumb2

Code 2.

sfhdr.stctlhdr.ctlmsgk

Message key used for the SNA sequence number. (For more information, see Remarks.)

Remarks

- The **Status-Control() ACKLUA** message consists of a buffer header only. There is no buffer element.
- The application flags and the code 1 and code 2 fields are undefined and should not be used.
- The message key field gives the sequence number from the transmission header of the inbound data message to which this is an acknowledgment.
- For a summary of **Status-Control** control type codes, see the table in Status-Control Message.

Status-Error

The **Status-Error** message is used to report request reject and response header (RH) usage error conditions in outbound SNA request/response units (RUs) to the application. It flows from the node to the application and is used with both system services control point (SSCP) and primary logical unit (PLU) connections.

For more information, see Status-Error Message.

Syntax

```
struct Status-Error {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.sterrhdr.errstat;
    CHAR      sfhdr.sterrhdr.errpad1;
    CHAR      sfhdr.sterrhdr.errpad2;
    CHAR      sfhdr.sterrhdr.errpad3;
    CHAR      sfhdr.sterrhdr.errcode1;
    CHAR      sfhdr.sterrhdr.errcode2;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.sterrhdr.errstat

Status type STERROR (0x03).

sfhdr.sterrhdr.errpad1

Reserved.

sfhdr.sterrhdr.errpad2

Reserved.

sfhdr.sterrhdr.errpad3

Reserved.

sfhdr.sterrhdr.errcode1

Error code 1.

sfhdr.sterrhdr.errcode2

Error code 2.

Remarks

- The **Status-Error** message consists of a buffer header only. There is no buffer element.
- The error codes are listed in Error and Sense Codes.

Status-Resource

The **Status-Resource** message is used to provide a simple flow control mechanism between the local node and the application to prevent the application from exhausting its resources. It flows from the application to the node, and is used with a primary logical unit (PLU) connection.

It is only used on the PLU connection where the application specifies in the PLU connection information control block (CICB) that pacing requires application participation. For further details, see Pacing and Chunking.

Syntax

```
struct Status-Resource {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.streshdr.resstat;
    CHAR      sfhdr.streshdr.respad;
    CHAR      sfhdr.streshdr.rescred;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.streshdr.resstat

Status type STRESRCE (0x04).

sfhdr.streshdr.respad

Reserved.

sfhdr.streshdr.rescred

Application credit.

Remarks

- The **Status-Resource** message consists of a buffer header only. There is no buffer element.
- The **rescred** (application credit) field indicates that the application can receive further credit request/response units (RUs) of the maximum RU size, or further credit chunks if chunking is being used.

Status-RTM

The **Status-RTM** message provides the application with information about the Response Time Monitor (RTM) measurement parameters used by the host. This allows the application to match its local display of RTM statistics, if it provides such a display, with the statistics used by the host. It flows from the node to the application and is used with an system services control point (SSCP) connection.

For further details, see Response Time Monitor Data.

Syntax

```
struct Status-RTM {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.strtmhdr.rtmstat;
    CHAR      sfhdr.strtmhdr.strbndry;
    CHAR      sfhdr.strtmhdr.strcount;
    CHAR      sfhdr.strtmhdr.strtmdef;
    CHAR      sfhdr.strtmhdr.strtmact;
    CHAR      sfhdr.strtmhdr.strtmdsp;
};
struct Status-RTM {
    PTRBFELT   hdreptr->elteptr;
    INTEGER    hdreptr->startd;
    INTEGER    hdreptr->endd;
    CHAR       hdreptr->trpad;
    CHAR[268]  hdreptr->dataru;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element.

numelts

Number of buffer elements.

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.strtmhdr.rtmstat

Status type STRTM (0x06).

sfhdr.strtmhdr.strbndry

RTM boundaries.

0x00 - No RTM boundaries follow in element. 0x01 - RTM boundaries follow in element.

sfhdr.strtmhdr.strcount

RTM counters.

0x00 - No RTM counters follow in element. 0x01 - RTM counters follow in element.

sfhdr.strtmhdr.strtmdef

RTM definition.

0x00 - No change: use last received definition. 0x01 - Timers run until first data is written to screen. 0x02 - Timers run until keyboard is unlocked. 0x03 - Timers run until application can send (change direction (CD) or end bracket (EB) received).

sfhdr.strtmhdr.strtmact

RTM measurement.

0x00 - not active 0x01 - active

sfhdr.strtmhdr.strtmdsp

Local RTM display.

0x00 - disabled 0x01 - enabled

Element

hdreptr–>elteptr

Pointer to buffer element (NIL).

hdreptr–>startd

Start of data in this element.

hdreptr–>endd

End of data in this element.

hdreptr–>trpad

Reserved.

hdreptr–>dataru

Data RU, as follows:

dataru[0–1]

Number of boundaries in element

0x0000 - no boundaries included

m - number of boundaries included

dataru[2–3]

Number of counters in element

0x0000 - no counters included

n - number of counters included

dataru[4–(2m+3)]

m boundary values.

dataru[(2m+4)–(2m+2n+3)]

*n*counter values.

dataru[(2m+2n+4)

RTM total time.

Remarks

- A **Status-RTM** message is sent after the **Open(SSCP) OK Response** to give the initial RTM parameters. It is sent again when the RTM counters are reset (either on request from the host or when the local node sends unsolicited RTM data to the host), or when the host changes any of the RTM parameters.

- The message is sent only for applications that use LUs with type video display unit (VDU) or logical units (LUs) in a VDU pool, because the RTM feature applies only to 3270 display sessions.

- All the values in the data RU are integer values.

- The RTM counter values in this message can be nonzero at startup, because RTM statistics are maintained for a specific LU and not for a specific application's use of that LU. If zero counter values are included, this indicates that the counters are to be reset.

- The RTM total time field is present only if the number of counters in the element is nonzero.

Status-Session

The **Status-Session** message provides the application with information about changes in the state of a session between the local node and the host. It flows from the node to the application and is used with both system services control point (SSCP) and primary logical unit (PLU) connections.

For further details, see Status-Session Message.

Syntax

```
struct Status-Session {
    PTRBFHDR  nxtqptr;
    PTRBFELT  hdreptr;
    CHAR      numelts;
    CHAR      msgtype;
    CHAR      srcl;
    CHAR      srcp;
    INTEGER   srci;
    CHAR      destl;
    CHAR      destp;
    INTEGER   desti;
    CHAR      sfhdr.stseshdr.sesstat;
    CHAR      sfhdr.stseshdr.sesspad;
    CHAR      sfhdr.steeshdr.sesscode;
    CHAR      sfhdr.stseshdr.sessqual;
};
```

Members

nxtqptr

Pointer to next buffer header.

hdreptr

Pointer to buffer element (NIL).

numelts

Number of buffer elements (0x00).

msgtype

Message type STATFMI (0x21).

srcl

Source locality.

srcp

Source partner.

srci

Source index.

destl

Destination locality.

destp

Destination partner.

desti

Destination index.

sfhdr.stseshdr.sesstat

Status type STSESSN (0x05).

sfhdr.stseshdr.sesspad

Reserved.

sfhdr.steeshdr.sesscode

Session code.

sfhdr.stseshdr.sessqual

Session code qualifier.

Remarks

- The **Status-Session** message consists of a buffer header only. There is no buffer element.
- The session codes are listed in Status-Session Codes.

FMI Extension for the Windows Environment

This section describes the application programming interface (API) extension to the Microsoft® Windows® 3270 Emulator Interface that converts link status and error codes to a printable string.

Following is a definition of the function, syntax, returns, and remarks for using the extension. For more information, see FMI Status, Error, and Sense Codes.

This section contains:

- GetFmiReturnCode

GetFmiReturnCode

The **GetFmiReturnCode** function converts link status and error codes to a printable string. This function provides a standard set of error strings for use by Function Management Interface (FMI) applications.

Syntax

```
int WINAPI GetFmiReturnCode (
    UINT errcode1,
    UINT errcode2,
    UINT buffer_length,
    unsigned char FAR *buffer_addr
);
```

Parameters

errcode1

Supplied parameter; see Remarks.

errcode2

Supplied parameter; see Remarks.

buffer_length

Supplied parameter; specifies the length of the buffer pointed to by *buffer_addr*. The recommended length is 256 characters.

buffer_addr

Supplied/returned parameter; specifies the address of the buffer that will hold the formatted, null-terminated string.

Return Values

0x20000001

The parameters are invalid; the function could not read the specified error codes or could not write to the specified buffer.

0x20000002

The specified buffer is too small.

Remarks

The *errcode1* and *errcode2* parameters are set according to the way that **GetFmiReturnCode** is used, as shown in the following table.

Codes to be translated	**Value for *errcode1***	**Value for *errcode2***
The *errcode1* and *errcode2* values specified in Error and Sense Codes includes messages for**Open(SSCP) Response**, **Open(PLU) Confirm**, **Status-Acknowledge(Nack-2)**, **Status-Control(...) Nack2**, **Status-Error**, and **Appl-Data** messages with the system detected error indicator (SDI) set	Unchanged from message	Unchanged from message
The status and qualifier codes returned from a Status-Session message	*status**256 + *qualifier*	0xFFFF
The return code from **WinLUAGetLastInitStatus**	The return code	0xFFFF

SNA Internationalization Programmer's Reference

This section describes the features available in Host Integration Server 2009 for programming support for international languages and different national language character sets.

The SNANLS API uses the language support features provided with Microsoft Windows Server 2003, Windows XP, and Windows 2000. SNANLS supports European languages that use single-byte encoding, as well as East Asian languages that use double-byte or Unicode encoding.

For general information about programming for SNA internationalization, see SNA Internationalization Programmer's Guide.

In This Section

SNANLS Code Page Support

The SNA National Language Support (SNANLS) API provides functions for converting single-byte character stream (SBCS) EBCDIC-to-Unicode-to-ANSI and SBCS ANSI-to-Unicode-to-EBCDIC by leveraging the Win32 National Language Support (NLS) API. The Win32 NLS API uses resource files containing NLS conversion tables that are installed on the target computer when Windows is installed or installed by the Setup program for Host Integration Server 2009 (the Setup program also adds the required registry entries). The SNANLS DLL is supplied with Host Integration Server.

SNANLS supports conversions for the following groups of code pages:

- ANSI code pages
- ANSI/OEM code pages
- EBCDIC code pages
- OEM PC code pages
- Open Systems code pages
- ISO code pages

In This Section

ANSI Code Page Support Using SNANLS

The following table shows the ANSI code pages and character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS display name	NLS code page	HOST CCSID	Type	NLS file name
ANSI - Arabic	1256	1256	SBCS	c_1256.nls
ANSI - Baltic	1257	1257	SBCS	c_1257.nls
ANSI - Cyrillic	1251	1251	SBCS	c_1251.nls
ANSI - Central Europe	1250	1250	SBCS	c_1250.nls
ANSI - Greek	1253	1253	SBCS	c_1253.nls
ANSI - Hebrew	1255	1255	SBCS	c_1258.nls
ANSI - Latin I	1252	1252	SBCS	c_1252.nls
ANSI - Turkish	1254	1254	SBCS	c_1254.nls

Note

All of these ANSI code pages support the euro.

ANSI/OEM Code Page Support Using SNANLS

The following table shows the ANSI/OEM code pages and character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS display name	NLS code page	Host CCSID	Type	NLS file name	Comments
ANSI/OEM - Japanese Shift-JIS	932	932	SBCS	c_932.nls	Japanese JIS-8 Bit + Shift-JIS
ANSI/OEM - Korean	949	949	DBCS	c_949.nls	Korean Hangul (Extended Wansung)
ANSI/OEM - Simplified Chinese GBK	936	936	DBCS	c_936.nls	Simplified Chinese GBK
ANSI/OEM - Thai	874	874	SBCS	c_874.nls	Thai
ANSI/OEM - Traditional Chinese Big5	950	950	DBCS	c_950.nls	Traditional Chinese Big5
ANSI/OEM - Viet Nam	1258	1258	SBCS	c_1258.nls	Viet Nam

Note

None of these code pages support the euro.

EBCDIC Code Page Support Using SNANLS

The following table shows the EBCDIC code pages and character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS Display Name	NLS Code Page	Host CCSID	euro	Supported by SNANLS
EBCDIC - Arabic	20420	420		partial (See Note)
EBCDIC - Cyrillic (Russian)	20880	880		yes
EBCDIC - Cyrillic (Serbian, Bulgarian)	21025	1025		yes
EBCDIC - Denmark/ Norway (Euro)	1142	277	yes	yes
EBCDIC - Denmark/ Norway	20277	277		yes
EBCDIC - Finland/ Sweden (Euro)	1143	278	yes	yes
EBCDIC - Finland/ Sweden	20278	278		yes
EBCDIC - France (Euro)	1147	297	yes	yes
EBCDIC - France	20297	297		yes
EBCDIC - Germany (Euro)	1141	273	yes	yes
EBCDIC - Germany	20273	273		yes
EBCDIC - Greek (Modern)	875	875		yes
EBCDIC - Greek	20423	423		yes
EBCDIC - Hebrew	20424	424		partial (See Note)
EBCDIC - Icelandic (Euro)	1149	871	yes	yes
EBCDIC - Icelandic	20871	871		yes
EBCDIC - International (Euro)	1148	500	yes	yes
EBCDIC - International	500	500		yes
EBCDIC - Italy (Euro)	1144	280	yes	yes
EBCDIC - Italy	20280	280		yes
EBCDIC - Japan English (Extended)	1027			
EBCDIC - Japan English/Kanji (Extended)	939	939		yes
EBCDIC - Japan English/Kanji (Extended)	5035			
EBCDIC - Japan Katakana (Extended)	290	290		yes

EBCDIC - Japan Katakana/Kanji (Extend Katakana)	930	930		yes
EBCDIC - Japan Katakana/Kanji (Extend Katakana)	5026			
EBCDIC - Japanese	931	931		yes
EBCDIC - Korea (Extended)	933	933		yes
EBCDIC - Latin America/ Spain (Euro)	1145	284	yes	yes
EBCDIC - Latin America/ Spain	20284	284		yes
EBCDIC - Multilingual/ ROECE (Latin-2)	870	870		yes
EBCDIC - Simplified Chinese (Extended)	935	935		yes
EBCDIC - Thai	20838	838		yes
EBCDIC - Traditional Chinese (Extended)	937	937		yes
EBCDIC - Turkish (Latin-3)	20905	905		yes
EBCDIC - Turkish (Latin-5)	1026	1026		yes
EBCDIC - U.S./ Canada (Euro)	1140	37	yes	yes
EBCDIC - U.S./ Canada	37	37		yes
EBCDIC - United Kingdom (Euro)	1146	285	yes	yes
EBCDIC - United Kingdom	20285	285		yes

Note

Support for Arabic and Hebrew code page conversions are limited to left-to-right output. Bidirectional output including the default Arabic and Hebrew right-to-left output is not supported in this release of Host Integration Server.

ISO Code Page Support Using SNANLS

The following table shows the ISO NLS code pages and host character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS display name	NLS code page	Host CCSID	euro	Supported by SNANLS
ISO 6937 Non-Spacing Accent	20269	6937		
ISO 8859-1 Latin-1	28591	819		
ISO 8859-15 Latin 9 (Euro)	20865	923	yes	
ISO 8859-2 Central Europe	28592	912		
ISO 8859-3 Latin 3	28593	913		
ISO 8859-4 Baltic	28594	914		
ISO 8859-5 Cyrillic	28595	915		
ISO 8859-6 Arabic	28596	1089		
ISO 8859-7 Greek	28597	813		
ISO 8859-8 Hebrew (Visually Ordered)	28598	916		
ISO 8859-9 Hebrew (Logically Ordered)	28599	920		

OEM PC Code Page Support Using SNANLS

The following table shows the OEM PC code pages and character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS display name	NLS code page	Host CCSID	Type	NLS file name	Comments
OEM - Arabic	864	864	SBCS	c_864.nls	
OEM - Baltic	775	775	SBCS	c_775.nls	
OEM - Canadian French	863	863	SBCS	c_863.nls	OEM - Canada (850 subset)
OEM - Cyrillic	855	855	SBCS	c_855.nls	
OEM - Cyrillic II	866	866	SBCS	c_866.nls	OEM - Russian
OEM - Greek 437G	737	737	SBCS	c_737.nls	
OEM - Hebrew	862	862	SBCS	c_862.nls	
OEM - Icelandic	861	861	SBCS	c_861.nls	OEM - Iceland
OEM - Modern Greek	869	869	SBCS	c_869.nls	
OEM - Multilingual Latin I	850	850	SBCS	c_850.nls	
OEM - Multilingual Latin II	852	852	SBCS	c_852.nls	
OEM - Nordic	865	865	SBCS	c_865.nls	OEM - Denmark, Norway, Finland, Sweden
OEM - Portuguese	860	860	SBCS	c_860.nls	OEM - Portugal (850 subset)
OEM - Turkish	857	857	SBCS	c_857.nls	
OEM - United States	437	437	SBCS	c_437.nls	

Note

None of these code pages support the euro.

Open Systems Code Page Support Using SNANLS

The following table shows the Open Systems NLS code pages and host character code set identifiers (CCSIDs) supported by SNA National Language Support (SNANLS) in Host Integration Server 2009.

SNANLS display name	NLS code page	Host CCSID	euro	Supported by SNANLS
Latin-1/Open System (Euro)	20924	924	yes	
Latin-1/Open System	1047	1047		

SNANLS Dependencies

The only file required to support the SNA National Language Support (SNANLS) API on Microsoft Windows Server 2003, Windows XP, and Windows 2000 is SNANLS.DLL. To link to this .dll, use the SNANLS.H header (located under the \SDK\INCLUDE subdirectory) and the SNANLS.LIB library file (located under the \SDK\LIB subdirectory) supplied with the Host Integration Server 2009 SDK. Note that individual Win32 NLS resource files must be installed in order to support the various languages and code pages on Windows Server 2003, Windows XP, and Windows 2000.

The Win32 NLS files needed to support various languages are normally installed when the operating system is installed during Setup for Windows Server 2003, Windows XP, and Windows 2000. If these files are not present on Windows Server 2003, Windows XP, and Windows 2000, they may be installed using Regional and Language Options. Click **Start**, then click **Control Panel**. Click **Regional and Language Options**, then click the **Advanced** tab. Select the appropriate settings from this dialog box.

The registry settings required to use specific NLS files are enabled on Windows Server 2003, Windows XP, and Windows 2000 when the operating system is installed. When you install the end-user client or Administrator clients from Host Integration Server 2009, the registry settings required to use specific NLS files are automatically created.

The registry settings required for common EBCDIC code pages are listed in the following table.

File name	SNANLS display name	NLS code page	Host CCSID	Registry setting
c_037.nls	EBCDIC - U.S./ Canada	37	37	Value Name=37 Type=REG_SZ Data=c_037.nls
c_500.nls	EBCDIC - International	500	500	Value Name=500 Type=REG_SZ Data=c_500.nls
c_870.nls	EBCDIC - Multilingual/ ROECE (Latin-2)	870	870	Value Name=870 Type=REG_SZ Data=c_870.nls
c_875.nls	EBCDIC - Greek (Modern)	875	875	Value Name=875 Type=REG_SZ Data=c_875.nls
c_1026.nls	EBCDIC - Turkish (Latin-5)	1026	1026	Value Name=1026 Type=REG_SZ Data=c_1026.nls
c_20273.nls	EBCDIC - Germany	20273	273	Value Name=20273 Type=REG_SZ Data=c_20273.nls
c_20277.nls	EBCDIC - Denmark/ Norway	20277	277	Value Name=20277 Type=REG_SZ Data=c_20277.nls
c_20278.nls	EBCDIC - Finland/ Sweden	20278	278	Value Name=20278 Type=REG_SZ Data=c_20278.nls
c_20280.nls	EBCDIC - Italy	20280	280	Value Name=20280 Type=REG_SZ Data=c_20280.nls
c_20284.nls	EBCDIC - Latin America/ Spain	20285	284	Value Name=20284 Type=REG_SZ Data=c_20284.nls
c_20285.nls	EBCDIC - United Kingdom	20285	285	Value Name=20285 Type=REG_SZ Data=c_20285.nls
c_20297.nls	EBCDIC - France	20297	297	Value Name=20297 Type=REG_SZ Data=c_20297.nls
c_20420.nls	EBCDIC - Arabic	20420	420	Value Name=28596 Type=REG_SZ Data=c_20420.nls

c_20423.nls	EBCDIC - Greek	20423	423	Value Name=20423 Type=REG_SZ Data=c_20423.nls
c_20424.nls	EBCDIC - Hebrew	20424	424	Value Name=20424 Type=REG_SZ Data=c_20424.nls
c_20838.nls	EBCDIC - Thai	20838	838	Value Name=20838 Type=REG_SZ Data=c_20838.nls
c_20871.nls	EBCDIC - Icelandic	20871	871	Value Name=20871 Type=REG_SZ Data=c_20871.nls
c_20880.nls	EBCDIC - Cyrillic (Russian)	20880	880	Value Name=20880 Type=REG_SZ Data=c_20880.nls
c_20905.nls	EBCDIC - Turkish (Latin-3)	20905	905	Value Name=20905 Type=REG_SZ Data=c_20905.nls
c_21025.nls	EBCDIC - Cyrillic (Serbian, Bulgarian)	21025	1025	Value Name=21025 Type=REG_SZ Data=c_21025.nls

Note

On Windows Server 2003, Windows XP, and Windows 2000, the registry settings are located under the **HKEY_LOCAL_MACHINE** under the following sub key: SYSTEM\CurrentControlSet\Control\Nls\CodePage.

SNANLS API Functions

The SNA National Language Support (SNANLS) API is documented in the SNANLS.H file in the software development kit (SDK) provided with Host Integration Server 2009.

SNANLS on Host Integration Server 2009 supports the following functions.

In This Section

- CloseNlsRegistry
- FindCloseCodePage
- FindFirstCodePage
- FindNextCodePage
- GetCodePage
- GetCodePageDisplayStr
- IsInstalledCodePage
- OpenNlsRegistry
- SnaNlsInit
- SnaNlsMapString

CloseNlsRegistry

The SNA National Language Support (SNANLS) **CloseNlsRegistry** function closes an open registry key on a local or remote computer.

Syntax

```
BOOL WINAPI CloseNlsRegistry(
HKEY KeyHandle
);
```

Parameters

KeyHandle

Supplied parameter. The handle to a key in the registry opened using **OpenNlsRegistry**.

Return Value

The **CloseNlsRegistry** function returns zero on success, otherwise a non-zero value is returned on failure.

Remarks

The *KeyHandle* parameter passed to this function is the handle returned from a previous call to the **OpenNlsRegistry** function. This function is primarily used by the Print service in Host Integration Server 2009 to determine what code pages are supported on a remote computer providing the print services function.

SNANLS supports this function on Microsoft Host Integration Server 2009.

FindCloseCodePage

The SNA National Language Support (SNANLS) **FindCloseCodePage** function closes the handle allocated by a call to the **FindFirstCodePage** function.

Syntax

```
BOOL WINAPI FindCloseCodePage(
        const HANDLEhInfo
);
```

Parameters

hInfo

Supplied parameter. The handle allocated and returned using **FindFirstCodePage**.

Return Value

The **FindCloseCodePage** function returns **TRUE** on success, otherwise the returned value on failure is **FALSE**.

Remarks

The *hInfo* parameter passed to this function is the handle returned from a previous call to the **FindFirstCodePage** function.

SNANLS supports this function on Host Integration Server 2009.

FindFirstCodePage

The SNA National Language Support (SNANLS) **FindFirstCodePage** function finds the first instance of a code page satisfying the condition specified, copies the code page information to a structure passed as a parameter, and opens and returns a handle used in subsequent calls to the **FindNextCodePage** function.

Syntax

```
const HANDLE WINAPI FindFirstCodePage(
        DWORDdwEnumOption,
        struct CodePage *pPage
);
```

Parameters

dwEnumOption

Supplied parameter. The set of conditions that a code page should satisfy. These conditions can be any combination of the following values defined in the SNANLS.h include file:

ENUM_CP_AVAILABLE (0x01)

The code page is installed and available for use.

ENUM_CP_HOST (0x02)

The code page is a host code page (EBCDIC, for example).

ENUM_CP_EURO (0x04)

The code page contains support for the euro character.

ENUM_CP_DBCS (0x08)

The code page is for a double-byte character set.

ENUM_CP_MBCS (0x10)

The code page is for a mixed-byte character set.

ENUM_CP_SBCS (0x20)

The code page is for a single-byte character set.

Note that some of these combinations represent cases that will not match any installed code pages used by SNANLS.

pPage

Supplied and returned parameter. A pointer to a struct CodePage where the code page information should be copied.

On a successful return, the memory location pointed to by this parameter will be filled with the information for the first code page satisfying the conditions in *dwEnumOption*. On failure, no changes will be made to the memory pointed to by this parameter.

The CodePage struct is defined in the SNANLS.H include file as follows:

```
struct CodePage {
    BYTE    CodePageKey;
    DWORD   CodePageID;
    WCHAR   szFriendlyName[CP_SIZE];
    short   eGroup;
    BOOL    bAvailable;
    BYTE    bccsid;
    BOOL    bEuro;
};
```

The members of this CodePage structure are as follows:

CodePageKey

A numeric value that represents the index into the array of CodePage structures. This value should be used as an opaque value, since this value can be changed arbitrarily by Service Packs when additional code pages are supported.

CodePageID

The NLS code page number.

szFriendlyName

The SNANLS display name for this code page.

eGroup

The group that this code page is represented by. .This value can be one of the following enumerations defined in the SNANLS.h include file for code groups:

ENUM_CP_EBCDIC

This code page is a member of the EBCDIC code page group.

ENUM_CP_ANSI

This code page is a member of the ANSI code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEMPC

This code page is a member of the OEM PC code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEM PC

This code page is a member of the OEM PC code page group.

ENUM_CP_OPEN

This code page is a member of the Open Systems code page group.

ENUM_CP_UCS

This code page is a member of the UCS code page group.

bAvailable

A Boolean used to indicate that this code page is installed on the computer. A value of **FALSE** for this member indicates that the computer will not be queried to determine if this code page is installed. A value of **TRUE** indicated the code page is installed.

bccsid

A flag used to indicate the type of code page. This flag can be one of the following:

ENUM_CP_DBCS (0x08)

The code page is for a double-byte character set.

ENUM_CP_MBCS (0x10)

The code page is for a mixed-byte character set.

ENUM_CP_SBCS (0x20)

The code page is for a single-byte character set.

bEuro

A Boolean value used to indicate if this code page supports the euro symbol. If this value is **TRUE**, then the euro symbol is

supported.

Return Value

The **FindFirstCodePage** function returns a handle used in calls to the **FindNextCodePage** or **FindCloseCodePage** on success. On failure, INVALID_HANDLE_VALUE is returned for the value of this handle.

Remarks

The handle returned by this function should not be tampered with by the user.

This function is supported by SNANLS on Host Integration Server.

FindNextCodePage

The SNA National Language Support (SNANLS) **FindNextCodePage** function finds the next instance of code page satisfying the condition specified in the initial call to the **FindFirstCodePage** function, and copies the code page information to a structure passed as a parameter.

Syntax

```
BOOL WINAPI FindNextCodePage(
        const HANDLE hInfo
struct CodePage *pPage
);
```

Parameters

hInfo

Supplied parameter. The handle allocated and returned using **FindFirstCodePage**.

pPage

Supplied and returned parameter. A pointer to struct CodePage where the code page information should be copied.

On a successful return, the memory location pointed to by this parameter will be filled with the information for the next code page satisfying the conditions in *dwEnumOption* parameter passed to the **FindFirstCodePage** function.

On failure, no changes will be made to the memory pointed to by this parameter.

The CodePage struct is defined in the SNANLS.H include file as follows:

```
struct CodePage {
    BYTE    CodePageKey;
    DWORD   CodePageID;
    WCHAR   szFriendlyName[CP_SIZE];
    short   eGroup;
    BOOL    bAvailable;
    BYTE    bccsid;
    BOOL    bEuro;
};
```

The members of this CodePage structure are as follows:

CodePageKey

A numeric value that represents the index into the array of CodePage structures. This value should be used as an opaque value, since this value can be changed arbitrarily by Service Packs when additional code pages are supported.

CodePageID

The NLS code page number.

szFriendlyName

The SNANLS display name for this code page. The character string is null terminated.

eGroup

The group that this code page is represented by. .This value can be one of the following enumerations defined in the SNANLS.h include file for code groups:

ENUM_CP_EBCDIC

This code page is a member of the EBCDIC code page group.

ENUM_CP_ANSI

This code page is a member of the ANSI code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEMPC

This code page is a member of the OEM PC code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEM PC

This code page is a member of the OEM PC code page group.

ENUM_CP_OPEN

This code page is a member of the Open Systems code page group.

ENUM_CP_UCS

This code page is a member of the UCS code page group.

bAvailable

A Boolean used to indicate that this code page is installed on the computer. A value of **FALSE** for this member indicates that the computer will not be queried to determine if this code page is installed. A value of **TRUE** indicated the code page is installed.

bccsid

A flag used to indicate the type of code page. This flag can be one of the following:

ENUM_CP_DBCS (0x08)

The code page is for a double-byte character set.

ENUM_CP_MBCS (0x10)

The code page is for a mixed-byte character set.

ENUM_CP_SBCS (0x20)

The code page is for a single-byte character set.

bEuro

A Boolean value used to indicate if this code page supports the euro symbol. If this value is **TRUE**, then the euro symbol is supported.

Return Value

The **FindNextCodePage** function returns a value of **TRUE** on success. On failure, the returned value is **FALSE**.

Remarks

This function is supported by SNANLS on Host Integration Server 2009.

GetCodePage

The SNA National Language Support (SNANLS) **GetCodePage** function copies the code page information identified by a key to a structure passed as a parameter.

Syntax

```
BOOL WINAPI GetCodePage(
        Int nKey
struct CodePage *pPage
);
```

Parameters

nKey

Supplied parameter. The numeric key to a code page. This value is an opaque index into an array containing the code pages supported by SNANLS. This value is normally the *CodePageKey* member of a CodePage structure returned from a previous call to **FindFirstCodePage** or **FindNextCodePage**.

pPage

Supplied and returned parameter. A pointer to struct CodePage where the code page information should be copied.

On a successful return, the memory location pointed to by this parameter will be filled with the information for the specific code page.

On failure, no changes will be made to the memory pointed to by this parameter.

The CodePage struct is defined in the SNANLS.H include file as follows:

```
struct CodePage {
    BYTE    CodePageKey;
    DWORD   CodePageID;
    WCHAR   szFriendlyName[CP_SIZE];
    short   eGroup;
    BOOL    bAvailable;
    BYTE    bccsid;
    BOOL    bEuro;
};
```

The members of this CodePage structure are as follows:

CodePageKey

A numeric value that represents the index into the array of CodePage structures. This value should be used as an opaque value, since this value can be changed arbitrarily by Service Packs when additional code pages are supported.

CodePageID

The NLS code page number.

szFriendlyName

The SNANLS display name for this code page. The character string is null terminated.

eGroup

The group that this code page is represented by. .This value can be one of the following enumerations defined in the SNANLS.h include file for code groups:

ENUM_CP_EBCDIC

This code page is a member of the EBCDIC code page group.

ENUM_CP_ANSI

This code page is a member of the ANSI code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEMPC

This code page is a member of the OEM PC code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_ISO

This code page is a member of the ISO code page group.

ENUM_CP_OEM PC

This code page is a member of the OEM PC code page group.

ENUM_CP_OPEN

This code page is a member of the Open Systems code page group.

ENUM_CP_UCS

This code page is a member of the UCS code page group.

bAvailable

A Boolean used to indicate that this code page is installed on the computer. A value of **FALSE** for this member indicates that the computer will not be queried to determine if this code page is installed. A value of **TRUE** indicated the code page is installed.

bccsid

A flag used to indicate the type of code page. This flag can be one of the following:

ENUM_CP_DBCS (0x08)

The code page is for a double-byte character set.

ENUM_CP_MBCS (0x10)

The code page is for a mixed-byte character set.

ENUM_CP_SBCS (0x20)

The code page is for a single-byte character set.

bEuro

A Boolean value used to indicate if this code page supports the Euro symbol. If this value is **TRUE**, then the euro symbol is supported.

Return Value

The **GetCodePage** function returns a value of **TRUE** on success. On failure, the returned value is **FALSE**.

Remarks

This function is supported by SNANLS on Host Integration Server 2009.

GetCodePageDisplayStr

The SNA National Language Support (SNANLS) **GetCodePageDisplayStr** function copies the SNANLS code page display name identified by a key to a character string passed as a parameter.

Syntax

```
BOOL WINAPI GetCodePageDisplayStr(
Int nKey
WCHAR *szDisplayStr
Int IDisplayStr
);
```

Parameters

nKey

Supplied parameter. The numeric key to a code page. This value is an opaque index into an array containing the code pages supported by SNANLS. This value is normally the *CodePageKey* member of a CodePage structure returned from a previous call to **FindFirstCodePage** or **FindNextCodePage**.

szDisplayStr

Supplied and returned parameter. A pointer to a wide-character array where the SNANLS display name for the specific code page should be copied.

On a successful return, the memory location pointed to by this parameter will be filled with the SNANLS display name for the specific code page. The character string is null terminated.

On failure, no changes will be made to the memory pointed to by this parameter.

IDisplayStr

Represents the number of available characters in the szDisplaystr parameter.

Return Value

The **GetCodePageDisplayStr** function returns a value of **TRUE** on success. On failure, the returned value is **FALSE**.

Remarks

This function is supported by SNANLS on Host Integration Server 2009.

IsInstalledCodePage

The SNA National Language Support (SNANLS) **IsInstalledCodePage** function determines if a code page is installed on a local or remote computer.

Syntax

```
BOOL WINAPI IsInstalledCodePage(
UINTCodePage,
HKEYKeyHandle
);
```

Parameters

CodePage

Supplied parameter. The NLS code page.

KeyHandle

Supplied parameter. The registry key returned from a call to the **OpenNlsRegistry** function.

Return Value

The **IsInstalledCodePage** function returns non-zero if a code page is installed, otherwise a zero value is returned on failure.

Remarks

This function is primarily used by the Print Service in Host Integration Server 2009 to determine if what code pages are supported on a remote computer providing the print services function.

This function is supported by SNANLS on Host Integration Server 2009.

OpenNlsRegistry

The SNA National Language Support (SNANLS) **OpenNlsRegistry** function opens a registry key on a local or remote computer pointing to the NLS Codepage registry entries.

Syntax

```
HKEY WINAPI OpenNlsRegistry(
char *MachineName,
HKEY hkey,
LPSTR Path
);
```

Parameters

MachineName

Supplied parameter. The name of the remote computer on which to open the registry. If this parameter is NULL, the registry on the local computer is opened.

hKey

Supplied parameter. The key to the registry to open. If this parameter is NULL, the **HKEY_LOCAL_MACHINE** key is used.

Path

Supplied parameter. The path to the key value in the registry hive to open. If this parameter is NULL, the following key is opened:

SYSTEM\CurrentControlSet\NLS\CodePage.

Return Value

The **OpenNlsRegistry** function returns a handle to the opened registry key on success, otherwise a NULL value is returned on failure.

Remarks

This function is primarily used by the Print Service in Host Integration Server 2009 to determine if what code pages are supported on a remote computer providing the print services function.

This function is supported by SNANLS in Host Integration Server 2009.

SnaNlsInit

The **SnaNlsInit** function is called to determine if the code page needed is supported by code page translations using SNA National Language Support (SNANLS). This enables an application to determine if the necessary NLS language files containing code page translation tables are installed on the local system.

Syntax

```
int WINAPI SnaNlsInit(
UINTCodePage
);
```

Parameters

CodePage

Supplied parameter. The number of the NLS code page for which support is requested. The *CodePage* parameter corresponds with the registry settings on Microsoft® Windows® 2000 located under the **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage** subkey.

Return Value

The **SnaNlsInit** function returns non-zero if code page translations are supported; otherwise 0 is returned.

Remarks

If CP_ACP (the current ANSI code page) is passed as the *CodePage* parameter, this functions returns non-zero.

This function is supported by SNANLS on Microsoft® Host Integration Server.

SnaNlsMapString

The **SnaNlsMapString** function is called to translate a string from one code page to another.

Syntax

```
int WINAPI SnaNlsMapString(
LPCTSTR lpSrcStr,
LPTSTR lpDestStr,
UINT inCodePage,
UINT outCodePage,
Int in_length,
int out_length,
UINT in_type,
UINT out_type,
WORD *Options,
LONG*lConvRequiredLen
);
```

Parameters

lpSrcStr

Supplied parameter. The input source string to be translated.

lpDestStr

Returned parameter. The translated string which may be NULL if *out_length* was zero.

inCodePage

Supplied parameter. Specifies the code page of the incoming source string; ignored if the input is Unicode.

outCodePage

Supplied parameter. Specifies the code page of the output translated string; ignored if the output is Unicode.

in_length

Supplied parameter. Specifies the length of the input source string in characters if the input is multibyte or in wide characters if the input is Unicode.

out_length

Supplied parameter. Specifies the maximum length available for the output translated string in characters if the output is multibyte or in wide characters if the output is Unicode.

in_type

Supplied parameter. Specifies the type of the input source string. Possible values for *in_type* are SNA_MULTIBYTE for multibyte and SNA_UNICODE for Unicode.

out_type

Supplied parameter. Specifies the type of the output translated string. Possible values for *out_type* are SNA_MULTIBYTE for multibyte and SNA_UNICODE for Unicode.

Options

Supplied and returned parameter. As a supplied parameter, this specifies a set of options that may be applied to the translation process, including TrnsDT options and the default character for the translation. On return, this parameter indicates the required buffer length for the output translated string if the function call failed.

Return Value

The **SnaNlsMapString** function returns the number of characters or wide characters written to *lpDestStr* on success; otherwise 0 is returned on failure.

On failure, the Win32® **GetLastError** function should be used to return an error code indicating the cause of the failure. Possible values returned by GetLastError are as follows:

ERROR_NOT_SUPPORTED

This error is returned for two possible reasons—either the NLS language resource file is not available or the *in_type* and *out_type* of the source and destination strings are not of the same type.

ERROR_BUFFER_OVERFLOW

This error is returned if the output buffer is too small. In such cases, the *Options* parameter returns with the value needed for *out_length*.

ERROR_INVALID_PARAMETER

This error is returned if a bad value was passed in a parameter; for example, if the *in_type* or *out_type* parameters contained undefined values.

ERROR_INVALID_DATA

This error is returned if a bad value was passed in the *lpSrcStr* parameter; for example, if the input string has a lead byte at the end.

ERROR_OUTOFMEMORY

This error is returned if memory could not be allocated for use by the SNANLS DLL.

TrnsDT API

The SNA National Language Support (SNANLS) API also enables applications to convert double-byte character stream (DBCS) EBCDIC-to-ANSI and DBCS ANSI-to-EBCDIC by leveraging another Host Integration Server 2009 API called TrnsDT. The TrnsDT API has its own mechanism to translate East Asia code pages using conversion table resource files (*.tbl files) that the Setup program for Host Integration Server 2009 installs on the target computer.

In This Section

TrnsDT Code Page Support

The TrnsDT API is used to perform all DBCS EBCDIC-to- ASCII conversions throughout Host Integration Server 2009. To a degree, TrnsDT has been and continues to be a uniform translation method and cross-component resource. TrnsDT also handles mixed DBCS and SBCS, plus SBCS for Japan.

In This Section

Host EBCDIC SBCS Using TrnsDT

The following table shows the character code set identifiers (CCSIDs) for EBCDIC single byte character sets (SBCS) supported by TrnsDT in Host Integration Server 2009.

Code page display name	Type	CCSID	Character set	Comments
IBM EBCDIC - U.S./Canada	SBCS	037	697	IBM English lowercase
IBM EBCDIC - Japan Katakana (Extended)	SBCS	290	1172	IBM Extended English Katakana
IBM EBCDIC - Korean (Extended)	SBCS	833	934	Korean (Extended)
IBM EBCDIC - Simplified Chinese (Extended)	SBCS	836	935	Simplified Chinese single-byte
IBM EBCDIC - Japan English (Extended)	SBCS	1027	1172	IBM Extended lowercase English
IBM EBCDIC - Traditional Chinese (Extended)	SBCS	28709	937	Traditional Chinese (Extended)

Host EBCDIC DBCS Using TrnsDT

The following table shows the character code set identifiers (CCSIDs) for EBCDIC double byte character sets (DBCS) supported by TrnsDT in Host Integration Server 2009.

Code page display name	Type	CCSID	Character set	Comments
IBM EBCDIC - Japan	DBCS	300	1001	IBM Japanese (including 4370 user-defined characters).
IBM EBCDIC - Korea	DBCS	834	933	IBM Korean (including 1880 user-defined characters).
IBM EBCDIC - Traditional Chinese	DBCS	835	937	Traditional Chinese Host double-byte (including 6204 user-defined characters)
IBM EBCDIC - Simplified Chinese	DBCS	837	837	Simplified Chinese Host double-byte
IBM EBCDIC - Japan	DBCS	4396	930, 931, 939	IBM Japanese (including 1880 user-defined characters).

Host EBCDIC Mixed SBCS and DBCS Using TrnsDT

The following table shows the character code set identifiers (CCSIDs) for EBCDIC mixed single byte character sets (SBCS) and double byte character sets (DBCS) supported by TrnsDT in Host Integration Server 2009.

Code page display name	Type	CCSID	Comments
IBM EBCDIC - Japan Katakana/Kanji (Extended)	Mixed	930	Japanese Katakana-Kanji mixed with 4370 user-defined characters.
IBM EBCDIC - Japanese	Mixed	931	Japan (English Lower-Case & Japanese)
IBM EBCDIC - Korea (Extended)	Mixed	933	Korean Mixed with 1880 user-defined characters.
IBM EBCDIC - Simplified Chinese (Extended)	Mixed	935	Simplified Chinese Host mixed with 1880 user-defined characters.
IBM EBCDIC - Traditional Chinese (Extended)	Mixed	937	Traditional Chinese Host mixed with 4370/6204 user-defined characters.
IBM EBCDIC - Japan English/Kanji (Extended)	Mixed	939	Japanese Latin Kanji mixed with 4370 user-defined characters.
IBM EBCDIC - Japan Katakana/Kanji (Extended)	Mixed	5026	A subset of CCSID 930 Japanese Katakana-Kanji mixed.
IBM EBCDIC - Japan English/Kanji (Extended)	Mixed	5035	A subset of CCSID 939 Japanese Latin Kanji mixed.

TrnsDT Conversions Possible

The following table describes conversions possible for the TrnsDT API.

Country/Region	Conversion	From CCSID	To CCSID
Japan	Host-PC	930	932
Japan	Host-PC	931	932
Japan	Host-PC	939	932
Japan	Host-PC	290	932
Japan	Host-PC	1027	932
Japan	Host-PC	5026	932
Japan	Host-PC	5035	932
Japan	PC-Host	932	930
Japan	PC-Host	932	931
Japan	PC-Host	932	939
Japan	PC-Host	932	290
Japan	PC-Host	932	1027
Japan	PC-Host	932	5026
Japan	PC-Host	932	5035
Taiwan	PC-Host	950	937
Taiwan	Host-PC	937	950
Korea	PC-Host	949	933
Korea	Host-PC	933	949
China	PC-Host	936	935
China	Host-PC	935	936

TrnsDT Resource Files

The **TrnsDt** API uses a series of resource files that contain the necessary translation tables, which are listed in the following table.

File name	Description
TRNSDT.DLL	Core global resource used by all TrnsDT conversions
TRNSDTJ.DLL	Core Japanese resource
TRNSDTS.DLL	Core Simplified Chinese (PRC) resource
TRNSDTK.DLL	Core Korean resource
TRNSDTT.DLL	Core Traditional Chinese (Taiwanese) resource
SNADBC.TBL	Japanese double-byte translation tables
SNADBCS.TBL	Simplified Chinese (PRC) double-byte translation tables
SNADBCT.TBL	Simplified Chinese (Taiwanese) double-byte translation tables
SNADBCK.TBL	Korean double-byte translation tables
SNASBC.TBL	Japanese single-byte translation tables
SNASBCS.TBL	Simplified Chinese (PRC) single-byte translation tables
SNASBCK.TBL	Korean single-byte translation tables
SNASBCT.TBL	Traditional Chinese (Taiwanese) single-byte translation tables

TrnsDT API Functions

The TrnsDT API consists of a single function.

In This Section

- TrnsDT
- PASSSTRUCT structure

TrnsDT

The **TrnsDT** function is called to translate a string from one code page to another.

Syntax

```
WORD WINAPI TrnsDt(
PASSSTRUCT far* PassParm);
```

Parameters

PassParm

Supplied parameter. A pointer to a **PASSSTRUCT** structure containing members that must be supplied as well as members that are returned by the function.

Return Value

The **TrnsDT** function returns zero on success. On failure, possible values returned by this function are as follows:

ERR_FILE_NOT_FOUND

This error is returned if the TrnsDT table files (*.tbl) could not be found. Normally **TrnsDT** uses the conversion tables located in the Host Integration Server\System directory on Microsoft® Windows Server™ 2003, Windows® XP, and Windows 2000. If **TrnsDT** cannot find these tables, it searches for them in the current directory.

ERR_INVALID_PARAMETER

This error is returned if a bad value was passed for one or more of the members of the *PassParm* structure. Invalid parameters can include not zeroing the **exit_code** member, passing an **in_length** for the input source string of zero or less or greater than 65535 bytes, passing an **out_length** for the output string buffer of zero or less, passing **in_page** or **out_page** members containing undefined codepage values.

ERR_BUFFER_OVERFLOW

This error is returned if the output buffer is too small for the converted output string. In such cases, the **out_length** member returns with the necessary value in bytes for the output buffer. This error is also returned if the length of the output buffer needed to convert the source string would exceed 65535 bytes.

ERR_MEMORY_ALLOCATE

This error is returned if memory could not be allocated for use by the TrnsDT DLL.

PASSSTRUCT structure

The PASSSTRUCT structure is defined as follows:

Syntax

```
typedef struct tagPassParm {
    WORD    parm_length;
    WORD    exit_code;
    WORD    in_length;
    LPBYTE  in_addr;
    WORD    out_length;
    LPBYTE  out_addr;
    WORD    trns_id;
    WORD    in_page;
    WORD    out_page;
    WORD    option;
} PASSSTRUCT;
```

Members

parm_length

Supplied parameter. The length of the structure passed, normally set to 24. If the **option** member is not needed or used, then this parameter can be set to 22.

exit_code

Supplied and returned parameter. On entry this member must be set to zero. On return, this member indicates the exit status. Legal values for returned **exit_code** values are as follows:

0

Normal exit code indicating function completed successfully.

1

The requested conversion is not supported.

12

The **exit_code** field was not properly initialized to zero.

128

The last character in the source input string was a DBCS lead byte.

256

The conversion could not be successfully completed since the length of the resulting converted destination string exceeds 65535 bytes.

257

An error occurred when trying to load one and initialize one of the TrnsDTx.dll files.

in_length

Supplied parameter. Specifies the length of the input source string in bytes.

in_addr

Supplied parameter. A pointer to the buffer containing the source string to be converted.

out_length

Supplied and returned parameter. Specifies the maximum length available for the output translated string in bytes. On return, this member is set to the length of the converted output string on success or the output buffer length needed if the buffer was too small.

out_addr

Supplied parameter. A pointer to the buffer that will contain the output destination string after conversion.

trns_id

Supplied parameter. The conversion identifier, which is always zero.

in_page

Supplied parameter. Specifies the code page of the incoming source string.

out_page

Supplied parameter. Specifies the code page of the output translated string.

option

Supplied and returned parameter if **parm_length** was set to 24. As a supplied parameter, this specifies a set of options that may be applied to the translation process. Possible values for these options are as follows:

Bits 15-9

Reserved.

Bit 8

Add shift out (SO)/shift in (SI) bytes to the converted output strings.

Bits 3-7

Reserved.

Bit 2

If this bit is set, then convert the input string using the IBM-specified 1-byte code table. This option is only valid when converting from code page 932 to one of the following code pages: 037, 290, 930, or 931.

If this bit is zero, then convert the input source string using the conversion table that is created using the SYSCTBL utility.

In case of double-byte characters, always use the conversion table created by the SYSCTBL utility.

The SYSCTBL.EXE file is a utility program included with Host Integration Server 2009 that provides a tool that can be used to create custom conversion tables for use with the **TrnsDT** function.

Bit 1

If this bit is set, then it indicates that the input source string starts with a 2-byte character. Generally, the host data always includes SO/SI control characters in pairs, but when converting part of mixed data strings, it is necessary to start the conversion from a double-byte character without the SO control character. In this case, the data itself does not have adequate information to determine if it is double-byte or not, so bit 1 must be set.

Bit 0

If this bit is set, then it indicates that the input source string contains SO/SI control characters. Bit 8 and bit 0 should be set as follows:

Conversion from PC to host Bit 8=1, bit 0 =0 Conversion from host to PC Bit 8=0, bit 0=1

On return, **option** is set to 4 if the last character was a double-byte character.

Host Integration Server Components and NLS Support

The following table lists the conversion methods and types used by the various components in Microsoft Host Integration Server 2009.

Component	SNANLS	TrnsDT (DBCS)	Bidirectional layout (Arabic)	Notes
3270 Applet	No	No	No	
5250 Applet	No	No	No	
Security Integration Service	Yes	Yes	No	
Host Security	No	No	No	User identifiers and passwords are converted from computer to host. Only Latin I code pages are supported.
NetView Alert Service	No	No	No	
NetView RunCmd Service	No	No	No	
CSV Convert Verb	No	No	No	
ODBC Driver for DB2	Yes	Yes	Yes	
OLE DB Provider for DB2	Yes	Yes	Yes	
OLE DB Provider for AS/400 and VSAM	Yes	Yes	Yes	
Managed Provider for DB2	Yes	Yes	Yes	
Data Queues ActiveX Control	Yes	Yes	No	
Host File Transfer ActiveX Control	Yes	Yes	No	
Transaction Integrator	Yes	Yes	Yes	

SNA Print Server Data Filter Programmer's Reference

The Host Print Service feature of Host Integration Server 2009 provides server-based 3270 and 5250 printer emulation, enabling host applications to print to a Local Area Network (LAN) printer supported by Microsoft Windows Server 2003, Windows XP, Windows 2000 Server, and Novell NetWare. This section introduces the SNA Print Server Data Filter API (sometimes referred to as the Print Exit API) that can be used to extend the capabilities of the Host Print Service in Host Integration Server.

This print data filter DLL can do the following:

- Send data to the printer when a job starts (print a banner page, for example).
- Perform special processing on the data to be printed.
- Send data to the printer upon print job completion (print a trailer page, for example).

In This Section

SNA Print Server Data Filter API

You configure the path to the print data filter DLL. This DLL is used by all sessions actively using the Host Print service. However, the print data filter DLL can specify whether or not it wants a given session's print data passed to it.

The entry points to this DLL are listed as follows:

PrtFilterAlloc
Obtains a data buffer in which to pass print data.

PrtFilterFree
Indicates that a data buffer obtained previously from the DLL is no longer needed and the DLL can free the memory allocated for this resource.

PrtFilterJobData
Allows the DLL to manipulate print data.

PrtFilterJobEnd
Informs the DLL that a print job has ended.

PrtFilterJobStart
Informs the DLL that a new print job has started and enables the DLL to send special data to the Print Server at the start of a job.

A description of the example sequence of calls during an ordinary print job is listed below to illustrate how these functions are normally used:

- **PrtFilterStartJob** is called when a new print job is started. The DLL can return a data buffer with special data that will be sent to the printer (a special banner page or special printer initialization strings, for example) before printing data.

- **PrtFilterFree** is called if special data was sent in the **PrtFilterStartJob** function and indicates that the data buffer used to pass special data can be freed.

The next sequence of function calls is repeated until all of the print data has been sent:

- **PrtFilterAlloc** is called to allocate a data buffer used to pass print data in the subsequent call to **PrtFilterJobData**.

- **PrtFilterJobData** is called to pass print data to the DLL for possible modification. This allows the user DLL the opportunity to manipulate the printer data before it is sent to the printer. If the modified print data to be returned requires a larger data buffer or the DLL needs to use a different data buffer for returning data, the DLL may need to allocate a new data buffer to return this data. The DLL may also choose to free the data buffer used to pass the incoming print data if a different data buffer is used to return modified print data. The **PrtFilterFree** function will not be called with the pointer to the original data buffer if a different data buffer is returned by **PrtFilterJobData**.

- **PrtFilterFree** is called to indicate that the data buffer allocated by **PrtFilterAlloc** for passing incoming data to the **PrtFilterJobData** function can be freed. If a different data buffer was returned by **PrtFilterJobData**, then **PrtFilterFree** would be called to indicate that a data buffer allocated by the DLL used to return modified print data in the **PrtFilterJobData** function can be freed.

The final sequence occurs when all of the print data has been processed:

- **PrtFilterEndJob** is called to indicate the end of the print job and allows the DLL the option to return special data (a trailer page, for example) that should be sent to the printer.

- **PrtFilterFree** is called if special data was sent in the **PrtFilterEndJob** function and indicates that the data buffer used to pass special data can be freed.

PrtFilterAlloc

The **PrtFilterAlloc** function is called to obtain a data buffer from the user filter DLL in which to pass it the print data.

Syntax

```
            void * WINAPI PrtFilterAlloc(
  DWORD BufLen
);
```

Parameters

BufLen

Supplied parameter. Indicates the length of buffer required.

Return Value

The **PrtFilterAlloc** function allocates a memory block of *BufLen* size and returns a pointer to the buffer. This function should return a NULL pointer on failure.

See Also

Reference

PrtFilterFree

PrtFilterFree

The **PrtFilterFree** function is called to indicate that a data buffer obtained previously from the DLL is no longer needed and the DLL can free the memory allocated for this resource. This function is called for data buffers returned from calls to **PrtFilterAlloc** as well as buffers that were allocated by the DLL to pass data in the **PrtFilterStartJob**, **PrtFilterJobData**, and **PrtFilterEndJob** functions.

Syntax

```
            void WINAPI PrtFilterFree(
  void *pBuf
);
```

Parameters

pBuf

Supplied parameter. Points to the data buffer that can be freed.

See Also

Reference

PrtFilterAlloc

PrtFilterJobData

The **PrtFilterJobData** function is called to give the user DLL the opportunity to manipulate the printer data before it is printed. This allows the DLL to provide custom processing for print data sent to the print server.

Syntax

```
          void WINAPI PrtFilterJobData(
    void *UniqueID,
    char **pBufPtr,
    DWORD *pBufLen  );
```

Parameters

UniqueID

Supplied parameter. The *UniqueID* value returned by the **PrtFilterJobStart** function to identify a print job.

pBufPtr

The print server passes the print data received from the host to the user DLL for processing in this incoming buffer. The user DLL returns to the print server a pointer to an outgoing buffer of data to be printed. This outgoing buffer pointer can be different from the received buffer pointer because the print data filter DLL can modify the data. Note that in this case **PrtFilterFree** will only be called by the Host Print Service for the outgoing buffer pointer. If necessary, the print data filter DLL must call its own free function on the incoming buffer pointer that was supplied to the **PrtFilterJobData** function. This incoming buffer was allocated by a Host Print Service by a previous call to **PrtFilterAlloc.**

pBufLen

Indicates the length of the data passed in the buffer to the print server and the length of the buffer returned to the print server by the user-provided DLL.

Remarks

The data in the buffer is printable ASCII and/or printer control sequences if these are being sent in the print jobs. The buffer returned by the user DLL does not have to be the same as the buffer passed in. The returned buffer will always be freed by calling **PrtFilterFree** after the data has been spooled. The unique identifier parameter *UniqueID* is the identifier returned from a previous call to the **PrtFilterJobStart** function.

See Also

Reference

PrtFilterFree

PrtFilterJobStart

PrtFilterJobEnd

The **PrtFilterJobEnd** function is called to inform the print data filter DLL that a print job is about to end. This allows the DLL to provide custom processing and send special data to the print server at the end of a print job.

Syntax

```
            void * WINAPI PrtFilterJobEnd(
  void *UniqueID,
  char **pBufPtr,
  DWORD *pBufLen
);
```

Parameters

UniqueID

Supplied parameter. The *UniqueID* value returned by the **PrtFilterJobStart** function to identify a print job.

pBufPtr

Returned parameter. Specifies a pointer to a buffer pointer holding additional data to be printed by the print server.

pBufLen

Returned parameter. Pointer to the length of the data provided by the print data filter DLL in the buffer.

Remarks

No data is passed in the buffer, but the user DLL can return print data which will be sent to the printer before the print job is ended.

See Also

Reference

PrtFilterJobStart

PrtFilterJobStart

The **PrtFilterJobStart** function is called to inform the print data filter DLL that a new job has just been started. This allows the DLL to provide custom processing and send special data to the print server at the beginning of a job.

Syntax

```
          void * WINAPI PrtFilterJobStart(
    char *SessionName,
    DWORD LUType,
    char **pBufPtr,
    DWORD *pBufLen      );
```

Parameters

SessionName

Supplied parameter. The name of the print session which has just started a print job. The *SessionName* is the same as that configured in using the SNA Print Service Admin tool.

LUType

Supplied parameter. Specifies the printer type. Valid values are LU 1, LU 3, or LU 6.2 printers, represented by an *LUType* value of 1, 3, or 6.

pBufPtr

Returned parameter. Specifies a pointer to a buffer pointer holding additional data to be printed by the print server.

pBufLen

Returned parameter. Pointer to the length of the data provided by the print data filter DLL in the buffer.

Return Value

The **PrtFilterJobStart** function returns a unique identifier (cast to a pointer to a void) if it wants the opportunity to filter the data for this print job.

If the user DLL returns a NULL pointer, it is indicating that it is not interested in filtering this job. No further calls to the user DLL will be made for this print job.

Remarks

No data is passed in the data buffer to the print data filter DLL in this call, but the DLL can return data in *pBufPtr* (for example, a banner page). The data returned from this call should be printable ASCII and/or printer control sequences.

SNADIS Drivers Programmer's Reference

This section provides reference material for developers writing their own SNALink software.

In This Section

Base/DMOD and SNALink Entry Points

This section gives definitions for Base/DMOD and SNALink entry points that must be supplied in an SNALink.

In This Section

SNAGetBuffer

The **SNAGetBuffer** function is called by an application to get a buffer with a requested number of elements.

```
PTRBFHDR SNAGetBuffer(
INTEGER numelts
);
```

Parameters
numelts

Number of elements required.

Return Values

A pointer to the buffer obtained. NULL if a buffer could not be obtained.

Remarks

Each element has a size of 268, the constant SNANBEDA in the header file SNA_DLC.H.

The returned buffer consists of a header and the required number of elements. The header points to the first element, which points to the next element and so on to make an element chain.

It is possible to add an element to an existing buffer by calling SNAGetElement to get the extra element. The new element should be added to the element chain of the buffer, and the number of elements count should be updated.

The application must release any buffers that are not transmitted.

SNAGetElement

The **SNAGetElement** function is called by an application to get a buffer element to append to an existing buffer.

```
VOID SNAGetElement(
PTRBFELT *eltptr
);
```

Parameters
eltptr

Pointer to a pointer to an element. On return, this points to a pointer to the element obtained, or to NULL if an element was not obtained (an internal error).

Remarks

This function should only be used to get extra elements for an existing buffer. SNAGetBuffer should be used to get a new buffer.

The new element should be added to the chain of elements from the existing buffer header and the count of the number of elements updated.

This function is typically used when a received buffer is being reused to transmit a message that is longer than the incoming message.

SNAGetLinkName

The SNALink can call the **SNAGetLinkName** function to obtain its configured SNALink name.

```
VOID SNAGetLinkName(
VCHAR *linkname
);
```

Parameters
linkname

A pointer to a buffer where the NULL-terminated SNALink name is stored.

Remarks

The buffer should be at least nine bytes in length.

SNAGetVersion

The **SNAGetVersion** function returns the major version number in the low byte and minor version number in the high byte.

```
VSHORT FAR SNAGetVersion(void);
```

SNALinkDispatchProc

The **SNALinkDispatchProc** function is the link dispatcher function. The Base calls this function whenever one of the following events occurs:

- A message arrives for the link.
- The Base timer expires.
- Contact is lost with the local node.

```
VOID SNALinkDispatchProc(
PTRBFNDR msgptr,
INTEGER function,
INTEGER locality
);
```

Parameters

msgptr

The message to be processed, or NULL if some other event is being notified.

function

The reason for **SNALinkDispatchProc** being called.

locality

L value (only valid for function SBLOST).

Remarks

The *function* parameter can have one of three values:

- 0—Message received.
- SBLOST—Contact lost with local node; L-value of locality.
- SBTICK—Base timer has expired; occurs every five seconds.

For suggested usage of this function, see Sample Code for SNALinkDispatchProc.

SNALinkInitialize

The **SNALinkInitialize** function initializes the SNALink. The Base calls this function when the SNALink is loaded into memory.

```
VOID SNALinkInitializer(
HANDLE event
);
```

Parameters
event

A handle to the global Base event.

Remarks

This function should:

- Read in required configuration information.
- Perform any required initialization of the hardware or device driver.
- Set up control blocks and data structures required internally by the SNALink.

SNALinkTerminate

The **SNALinkTerminate** function terminates the SNALink. The Base calls this function, when present, during service shutdown. This allows the DLL to free memory, release system resources (such as events), and close drivers.

```
VOID SNALinkTerminate(void);
```

Remarks

This function must not send messages to other SNA components.

SNALinkWorkProc

The **SNALinkWorkProc** function is the work manager function. The Base calls this function whenever the global Base event is triggered by the SNALink, or at least once every five seconds.

```
VOID SNALinkWorkProc(void);
```

Remarks

This function can be used to perform any general processing required by the SNALink, in particular to process messages received from the link.

SNAReleaseBuffer

The **SNAReleaseBuffer** function is called by an application to release a buffer.

```
VOID SNAReleaseBuffer(
PTRBFHDR msgptr
);
```

Parameters
msgptr

Pointer to the buffer to be released.

Remarks

It is important that buffers are released after use. This is done automatically when a message is transmitted. For messages received, it is the responsibility of the application either to release or to reuse the buffer.

This function releases both the buffer header and any associated buffer elements. It is possible to release single elements from a buffer by using the function SNAReleaseElement.

SNAReleaseElement

The **SNAReleaseElement** function is called by an application to release a buffer element from an existing buffer.

```
VOID SNAReleaseElement(
PTRBFELT *eltptr
);
```

Parameters
eltptr

Pointer to a pointer to the element to be released.

Remarks

This function should only be used to release surplus elements from a buffer. SNAReleaseBuffer should be called to release the entire buffer.

The released element should first be removed from the element chain and the count of the number of elements updated.

This function is typically used when a received buffer is being reused to transmit a message that is shorter than the incoming message.

SNASendAlert

The SNALink calls the **SNASendAlert** function to send a complete preformatted Network Management Vector Transport (NMVT) alert to NetView.

```
VOID SNASendAlert(
PTRBFHDR msgptr,
INTEGER severity
);
```

Parameters

msgptr

Pointer to the NMVT alert to be sent.

severity

The severity of the problem that caused the alert (ranges from 0 through 16).

Remarks

The complete NMVT to be sent must be generated by the SNALink and inserted into a buffer. Only the elements are used—the buffer header need not be set up before sending. The fields **startd** and **endd** should be set to reflect the location of the NMVT within the element. Multiple elements can be used to store the NMVT, up to a maximum length of 512 bytes. The buffer will be freed by the Base after the NMVT has been sent.

Any NMVT sent refers to a particular Host Integration Server 2009 connection. It is recommended that the NMVT include at least a hierarchy resource list, giving the name of the remote physical unit (PU) that the connection is associated with. This name is supplied to the SNALink on the Open(STATION) message.

For complete details of the format of an NMVT, see the IBM manual *SNA Formats* (GA27-3136).

SNASendMessage

The **SNASendMessage** function is called by an application to send messages to other localities (in the case of an SNALink, the local 2.1 node).

```
VOID SNASendMessage(
PTRBFHDR *msgptr
);
```

Parameters
msgptr

Pointer to message to be sent.

Remarks

The locality, partner, index (LPI) values on the message should be set up to reference the correct connection for the data to be passed.

SNADIS Message Formats

This section describes the SNA Device Interface Specification interface in terms of message formats. These are presented in a language-independent notation that is described below.

The messages used between the node and the SNALinks are shown in the following table.

Message type	Direction	LPI connection
Open(LINK)Request	NODE ------> DLC	LINK
Close(LINK)Request	NODE ------> DLC	LINK
Send-XID	NODE ------> DLC	LINK
Open(STATION) Request	NODE ------> DLC	STATION
Close(STATION) Request	NODE ------> DLC	STATION
Open(LINK) Response	NODE <------ DLC	LINK
Close(LINK) Response	NODE <------ DLC	LINK
Request-Open-Station	NODE <------ DLC	LINK
Open(STATION) Response	NODE <------ DLC	STATION
Close(STATION) Response	NODE <------ DLC	STATION
Station-Contacted	NODE <------ DLC	STATION
Outage	NODE <------ DLC	LINK/STATION
DLC-Data	NODE <-----> DLC	STATION
Status-Resource	NODE <-----> DLC	STATION

Details of the message format notation and key assumptions about the contents of the message formats are as follows:

- "Reserved" indicates that the field must be set to zero (for a numeric field) or all nulls (for names) by the sender of the message.
- "Undefined" indicates that the value of the field is indeterminate. The field is not set by the sender and should not be examined by the receiver of the message.
- Fields that occupy two bytes — the **srci** field in all messages, and fields such as **opresid** in Open(LINK) Request — are represented with the arithmetically most significant byte in the lowest byte address, irrespective of the normal byte order used by the processor on which the software executes. That is, the 2-byte value 0x1234 has the byte 0x12 in the lowest byte address. The exception to this is the **startd** and **endd** fields in all elements, which are always stored in the processor's normal byte order.
- Messages are composed of buffers, consisting of a buffer header and zero or more buffer elements. For more information on buffer formats, see Messages.
- The **startd** field in each element gives the offset of the first byte of data in the element after the **trpad** field. Its value will either be 1 (data starts in the byte after the **trpad** field), 10 (nine bytes of padding are included between the **trpad** field

and the start of the data), or 13 (12 bytes of padding are included between the **trpad** field and the start of the data). Any extra bytes are used by the local node for additional header information. This avoids having to copy data into a new buffer when adding this information.

- Because **startd** indicates the index into **dataru** starting from 1, not 0, the first byte of valid data will always be at **dataru[startd–1]**.

- All fields within **dataru** are of type unsigned character (UCHAR), except where the notes indicate otherwise.

In This Section

Open(LINK)

Open(LINK) is used by the node to open the LINK LPI connection.

Open(LINK) Request

Flow : NODE ------> DLC

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue.
hdreptr	PTRBFELT	Pointer to first buffer element.
numelts	CHAR	Number of buffer elements: 1 (Number of elements can be 2 if the connection is for an X.25 SVC).
msgtype	CHAR	Message type: OPENMSG (0x01).
srcl	CHAR	Source locality.
srcp	CHAR	Source partner.
srci	INTEGER	Source index.
destl	CHAR	Destination locality.
destp	CHAR	Destination partner.
desti	INTEGER	Destination index.
ophdr.openqual	CHAR	Open qualifier: REQU (0x01).
ophdr.opentype	CHAR	Open type: LINK (0x10).
ophdr.opresid	INTEGER	Resource identifier.

Element 1

Field	Type	Description
hdreptr–>elteptr	PTRBFELT	Pointer to optional second buffer element (NULL if only one element).
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array.
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array.
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+9]	Not applicable	Source name — name of local node.
dataru[s+10..s+19]	Not applicable	Destination name—name of remote PU (blank for incoming calls).
dataru[s+20..s+21]	Not applicable	Link index.

Link Data (depends on DLC type)

Unless otherwise stated, these fields are valid for outgoing calls only.

SDLC link data field	Description
dataru[s+22]	XID supplied 0x00 Do not send initial XID 0x01 Send initial XID from this message (may be a NULL XID)
dataru[s+23]	Link operational role 0x00 Primary 0x01 Secondary 0x02 Negotiable
dataru[s+24]	Use Reject_Command indicator 0x00 Do use it 0x01 Do not use it
dataru[s+25]	Address match byte 0x00 Primary/Negotiable SDLC 0x01 to 0xFE Secondary SDLC
dataru[s+26]	Second SDLC address match byte 0x00 Primary SDLC 0xFF Secondary/Negotiable SDLC
dataru[s+27]	Reserved

Channel adapter link data	Description
dataru[s+22]	XID supplied. 0x00 Do not send initial XID. 0x01 Send initial XID from this message (may be a NULL XID).
dataru[s+23]	PU emulation type. 0x00 Unknown. 0x20 PU 2.0 (format 0 XID). 0x21 PU 2.1 (format 3 XID).
dataru[s+24]	Control Unit Image number (0 to 15) on the Channel address configured in SNA Manager.
dataru[s+25]	Channel subaddress.
dataru[s+26..s+67]	Reserved (may not be zero).

Station timers	Description
dataru[s+28..s+29]	Contact time-out.
dataru[s+30..s+31]	Contact retry limit.
dataru[s+32..s+33]	Discontact time-out.
dataru[s+34..s+35]	Discontact retry limit.
dataru[s+36..s+37]	Negative poll time-out.
dataru[s+38..s+39]	Negative poll retry limit.
dataru[s+40..s+41]	T1 (no acknowledgment) time-out.
dataru[s+42..s+43]	T2 (acknowledgment) time-out.
dataru[s+44..s+45]	Remote station busy time-out.
dataru[s+46..s+47]	Remote station busy retry limit.

Link timers	Description
dataru[s+48..s+49]	Idle time-out.

dataru [s+50.. s+51]	Idle retry limit.
dataru [s+52.. s+53]	Nonproductive receive time-out.
dataru [s+54.. s+55]	Nonproductive receive retry limit.
dataru [s+56.. s+57]	Write time-out.
dataru [s+58.. s+59]	Write retry limit.
dataru [s+60.. s+61]	Link connection time-out.
dataru [s+62.. s+63]	Link connection retry limit. 0xFFFF for infinite retry.
dataru [s+64.. s+65]	Reserved.
dataru [s+66]	Configuration options: Bit 0 : 1 = Constant carrier selected Bit 1 : 1 = NRZI 0 = NRZ Bit 2 : = Reserved Bit 3 : 1 = Full-duplex 0 = Half-duplex Bit 4 : 0 = External clocking Bit 5 : 1 = Data signal rate select high 0 = Data signal rate select low Bit 6 : 1 = Select standby on 0 = Select standby off Bit 7 : = Reserved
dataru [s+67]	Configuration options: line type 0x00 leased 0x01 switched manual dial 0x02 switched auto-dial

Note that for configuration options, bit 0 is the most significant option and bit 7 is the least significant. Reserved bits are not always zero, so always use a bitwise **AND** operation when testing these bits.

The configuration options byte is also valid for incoming calls.

In This Section

- Expanded Information About Message Formats for Open(LINK) Request with SDLC
- Optional Second Element (Only Used by X.25 SVC)

Expanded Information About Message Formats for Open(LINK) Request with SDLC

The following list supplements the information found in the table in Open(LINK) Request. The timers described in the lists are used by a synchronous data link control (SDLC) link service to determine when to retry communication and when to generate outages. Generally, after the time interval specified by the time-out (usually 1000 milliseconds), the communication is retried. The cycle of time-out and retry is repeated until the retry limit is reached. Then an Outage message is sent by the SDLC link service.

With some timers, there are no communication retries. Such timers simply cycle through the time-out as many times as allowed in the retry limit (without actually retrying), then generate an **Outage** message.

Each description indicates whether you can configure the field through a Microsoft Host Integration Server 2009 interface (such as the SNA Manager program). If the field is not configurable, the built-in setting for the field is shown.

SDLC Link Data

- *dataru[s+22] XID supplied*

  This field controls whether an initial exchange identification (XID) is sent on this connection. The value used is determined by the leased or switched setting for the line:

  Leased line: 0x00 Do not send an initial XID. Switched line: 0x01 Send an initial XID (may be a NULL XID).

  A line is configured as leased or switched in Host Integration Server Setup.

- *dataru[s+24] Use Reject_Command indicator*

  This field determines that the link service will not send a Reject command (an SDLC command, not often used, value 0x19) if a frame is received with an invalid next-to-send (NS) value. Instead, the link service waits until the next poll before requesting retransmission of the frame.

  This field is not configurable and must remain at the setting of "Do not use."

- *Station Timers (described for SDLC only)*

- dataru[s+28..s+29] Contact time-out

- dataru[s+30..s+31] Contact retry limit

  This timer is started when an XID or set normal response mode (SNRM) is transmitted. If the time-out expires without acknowledgment, the frame is retransmitted. When the number of retransmitted frames reaches the retry limit, an outage is generated. Note that for XIDs, the time-out value is randomized to prevent possible clashes between two servers sending XIDs simultaneously.

  This timer is configurable in SNA Manager, in the advanced settings for an SDLC connection.

- *dataru[s+32..s+33] Discontact time-out*

- *dataru[s+34..s+35] Discontact retry limit*

  This timer is started when a discontact (DISC) is sent. It is stopped when an unnumbered acknowledgment (UA) or disconnect mode (DM) is received. If the number of sent DISCs reaches the retry limit, an outage is generated.

  This timer is not configurable. The discontact time-out is 1000 milliseconds; the discontact retry limit is 3.

- *dataru[s+36..s+37] Negative poll time-out*

- *dataru[s+38..s+39] Negative poll retry limit*

  This timer is used for primary SDLC only. At intervals specified by the negative poll time-out, a receive ready (RR) is transmitted. The negative poll retry limit is set at no limit; therefore, no outage is generated, no matter how many RRs are transmitted without acknowledgment being received.

  The negative poll time-out is configurable in SNA Manager, in the advanced settings for an SDLC connection, where the time-out is called poll rate. Poll rate is set in polls per second (and translated internally into the negative poll time-out, timed in milliseconds).

  The negative poll retry limit is not configurable. It is set at –1, meaning no limit.

- *dataru[s+40..s+41] T1 (no acknowledgment) time-out*

- *dataru[s+42..s+43] N2 (no acknowledgment) retry limit*

  This timer is used for primary SDLC and is started when a poll/final bit is expected. If the time-out expires before a frame containing a poll/final bit is received, an RR is sent. When the number of sent RRs reaches the retry limit, an outage is generated.

  This timer is configurable in SNA Manager, in the advanced settings for an SDLC connection, where it is called the poll time-out and poll retry limit.

- *dataru[s+44..s+45] Remote station busy time-out*

- *dataru[s+46..s+47] Remote station busy retry limit*

  This timer is used for primary SDLC and is started when a receive not ready (RNR) is received. It is stopped when an RR is received. If the time-out expires the number of times specified by the retry limit, an outage is generated.

  This timer is not configurable. The remote station busy time-out is 1000 milliseconds. The remote station busy retry limit is 30. Therefore, the time allowed before an outage is 30 seconds.

Link Timers (described for SDLC only)

- *dataru[s+48..s+49] Idle time-out*

- *dataru[s+50..s+51] Idle retry limit*

  This timer is configurable in SNA Manager, in the advanced settings for an SDLC connection.

- *dataru[s+52..s+53] Nonproductive receive time-out*

- *dataru[s+54..s+55] Nonproductive receive retry limit*

  This timer is used for secondary SDLC only and is started when any frame is received for this station. It is stopped when additional frames are received for this station. If the time-out expires the number of times specified by the retry limit, the link service causes a pop-up message, but does not generate an outage (because multidrop lines can be very slow).

  This timer is not configurable. The nonproductive receive time-out is 1000 milliseconds (1 second); the nonproductive receive retry limit is 60. Therefore, the time allowed before a pop-up message is 60 seconds.

- *dataru[s+56..s+57] Write time-out*

- *dataru[s+58..s+59] Write retry limit*

  This timer is started after an information frame has been transmitted to the hardware and stopped when the hardware

acknowledges the frame. If the time-out expires the number of times specified by the retry limit, an outage is generated.

This timer is not configurable. The write time-out is 1000 milliseconds (one second). The write retry limit is 15. Therefore, the time allowed before an outage is 15 seconds.

- *dataru[s+60..s+61] Link connection time-out*

- *dataru[s+62..s+63] Link connection retry limit*

This timer is started when an open link for a leased line is received, and stopped when Data Set Ready (DSR) is raised. If the time-out expires the number of times specified by the retry limit, an outage is generated.

This timer is not configurable. The link connection time-out is 1000 milliseconds (one second). The link connection retry limit is 300. Therefore, the time allowed before an outage is 300 seconds.

X.25 link data	**Description**
dataru[s+22]	Circuit type 0x00 PVC 0x01 SVC
dataru[s+23]	PVC alias, starting at 1 for lowest PVC channel number (reserved for SVC).
dataru[s+24..s+25]	PVC packet size (reserved for SVC).
dataru[s+26]	Default level 3 window size for PVC (reserved for SVC).
dataru[s+27]	Link role: 0x00 Primary 0x01 Secondary 0x02 Negotiable

802.2 link data	**Description**
dataru[s+22]	Maximum receives without a transmit acknowledgment.
dataru[s+23]	Maximum transmits without a receive acknowledgment.
dataru[s+24]	Dynamic window increment value.
dataru[s+25]	Remote local service access point (SAP) address.
dataru[s+26]	Local SAP address (for incoming calls).
dataru[s+27]	Value for t1 timer multiplier.
dataru[s+31]	Value for t2 timer multiplier.
dataru[s+35]	Value for t3 timer multiplier.
dataru[s+42]	Maximum retry count (N2 value). Note that this 802.2 link data is required for the 802.2 command DLC.OPEN.STATION.

Note that if the fields in the preceding table for timer multipliers are set to zero, the SNALink should use appropriate defaults.

End of link data section	**Description**
dataru[s+68..s+69]	Length of link connection data (= a) (0x0000) None present
dataru[s+70..s+70+a]	Link connection data

dataru[s+70+b..s+71+b]	Where b is maximum of a and 20. Size of XID I-frame (= n) (0x0000) NULL XID
dataru[s+72+b]	XID

Note that if there are 20 or fewer bytes of link connection data, the XID length is at s+90 and the actual XID starts at s+92.

The link connection data can contain one of the following:

- Media access control (MAC) address of remote station
- X.25 address of remote station
- Dial-digits for manual or auto-dial modems

Optional Second Element (Only Used by X.25 SVC)

Element 2

Field	Type	Description
hdreptr–>elteptr–>eltept r	PTRBFELT	Pointer to next buffer element: NULL
hdreptr–>elteptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>elteptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>elteptr–>datar u	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s]	Not applicable	Length of facilities data field (= c) inclusive of this length byte 0x01 no facilities d ata
dataru[s+1..s+c–1]	Not applicable	CHAR[c–1]Facilities data
dataru[s+c]	Not applicable	CHARLength of user data field (= d) inclusive of this length byte 0x01 no user dat a
dataru[s+c+1..s+c+d]	Not applicable	User data

Open(LINK) Response

Flow : DLC ------> NODE

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element
numelts	CHAR	Number of buffer elements: 1
msgtype	CHAR	Message type: OPENMSG (0x01)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
ophdr.openqual	CHAR	Open qualifier - RSPOK (0x02) - RSPERR (0x03)
ophdr.opentype	CHAR	Open type - LINK (0x10)
ophdr.opresid	INTEGER	Resource identifier
ophdr.oprr1	INTEGER	Error code (see immediately following the table)
ophdr.oprr2	INTEGER	Reserved
hdreptr–>elteptr	PTRBFELT	Pointer to next buffer element: NULL
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+9]	Not applicable	Source name—same as destination name from **Open(LINK) Request**
dataru[s+10..s+19]	Not applicable	Destination name—name of local node; same as source name from **Open(LINK) Request**

dataru[s+22..s+23]	Not applicable	The maximum BTU size supported by SNALink. This size is 65,536 (largest number in an unsigned short) for channel connections and 32,768 for non-channel connections. Note that this limit does not guarantee that the SNA connection will actually use this value. The individual link service or the host can negotiate it downward.

The error codes (for an ERROR-RESPONSE) are defined as follows in SNA_CNST.H:

Symbolic constant	Value	Description
ERINIFAIL	0x01	Hardware initialization failed
ERINVXID	0x08	Invalid XID length
ERLINKOPN	0x09	Link already open
ERLLCERR	0x0A	LCC error; fatal hardware failure
ERBADINDX	0x0B	Invalid link index
ERBADOPN	0x0C	**Open(LINK)** has insufficient data
ERCONNTO	0x0D	Link connection time-out
ERNORES	0x0E	Maximum connection count reached –or– No more internal control blocks
EROPNPND	0x11	**Close(LINK)** arrived while **Open(LINK)** pending
ERDUPREQ	0x12	Duplicate request

Close(LINK)

Close(LINK) is used by the node to close the LINK LPI connection.

Close(LINK) Request

Flow : NODE ------> DLC

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: CLOSEMSG (0x02)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
clhdr.closqual	CHAR	Close qualifier: REQU (0x01)
clhdr.clstype	CHAR	Close type: LINK (0x10)

Note that the message consists of a buffer header only.

Close(LINK) Response

Flow : DLC ------> NODE

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: CLOSEMSG (0x02)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
clhdr.closqual	CHAR	Close qualifier - RSPOK (0x02) - RSPERR (0x03)
clhdr.clstype	CHAR	Close type: LINK (0x10)
clhdr.clserr1	INTEGER	Error code (see immediately following the table)

The error codes (for an ERROR-RESPONSE) are defined as:

- 0x03 — Link not open
- 0x04 — Invalid link index

Note
The **Close(LINK)** message unconditionally shuts down the link.

Open(STATION)

Open(STATION) is used by the node to open the STATION LPI connection.

Open(STATION) Request

Flow : NODE ------> DLC

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element
numelts	CHAR	Number of buffer elements: 1
msgtype	CHAR	Message type: OPENMSG (0x01)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
ophdr.openqual	CHAR	Open qualifier: REQU (0x01)
ophdr.opentype	CHAR	Open type: STAT (0x11)
ophdr.opresid	INTEGER	Resource identifier
ophdr.icreditr	INTEGER	Initial credit for flow DLC –> NODE

Element

Field	Type	Description
hdreptr–>elteptr	PTRBFELT	Pointer to optional second buffer element (NULL if only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+9]	Not applicable	Source name—name of local node
dataru[s+10..s+19]	Not applicable	Destination name
dataru[s+20..s+21]	Not applicable	Link index as specified in **Open(LINK) Request**
dataru[s+22]	Not applicable	If NODE is primary, address of secondary station to initiate contact procedure with. 0x00 if NODE is secondary

dataru[s+23]	Not applicable	FID2 indicator 0x00 FID2 used
dataru[s+24]	Not applicable	Station type 0x00 Subarea 0x01 Peer
dataru[s+25..s+26]	Not applicable	Length of network name from received XID 0000 = No name
dataru[s+27..s+27+n]	Not applicable	Network name from received XID, in local character set, or if this is null, the name of the remote PU record in the COM.CFG file. This name can be fully qualified and has a maximum length of 17 characters.
dataru[s+44..s+89]	Not applicable	Link data—a copy of that supplied on the **Open(LINK) Request**
dataru[s+90..s+91]	Not applicable	The maximum BTU size to be used with this station. This size is 65,536 (largest number in an unsigned short) for channel connections and 32,768 for non-channel connections. Note that this limit does not guarantee that the SNA connection will actually use this value. The individual link service or the host can negotiate it downward.
dataru[s+m+1]	Not applicable	Local service access point (SAP) used by remote station. The remote SAP information is only allowed for 802.2 connections, and may only be present if Signaling information is present. It is used along with the Signaling Information to identify the remote station.

Open(STATION) OResponse

Flow : DLC ------> NODE

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element
numelts	CHAR	Number of buffer elements: 1
msgtype	CHAR	Message type: OPENMSG (0x01)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
ophdr.openqual	CHAR	Open qualifier: RSPOK (0x02)
ophdr.opentype	CHAR	Open type: STAT (0x11)
ophdr.opresid	INTEGER	Resource identifier
ophdr.icreditr	INTEGER	Initial Credit for flow DLC –> NODE
ophdr.icredits	INTEGER	Initial Credit for flow NODE –> DLC

Element

Field	Type	Description
hdreptr–>elteptr	PTRBFELT	Pointer to next buffer element (NULL is only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+9]	Not applicable	Source name—same as destination name from **Open(STATION) Request**
dataru[s+10..s+19]	Not applicable	Destination name—name of local node; same as source name from **Open(STATION) Request**

Open(STATION) Error Response

Flow : DLC ------> NODE

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element
numelts	CHAR	Number of buffer elements: 1
msgtype	CHAR	Message type: OPENMSG (0x01)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
ophdr.openqual	CHAR	Open qualifier: RSPERR (0x03)
ophdr.opentype	CHAR	Open type: STAT (0x11)
ophdr.opresid	INTEGER	Resource identifier
ophdr.operr1	INTEGER	Error code
ophdr.operr2	INTEGER	Reserved

Element

Field	Type	Description
hdreptr–>elteptr	PTRBFELT	Pointer to next buffer element (NULL is only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array - 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd - 1
dataru[s..s+9]	Not applicable	Source name
dataru[s+10..s+19]	Not applicable	Destination name

The error codes are defined as follows:

Symbolic constant	Value	Description

ERLKNOTOPEN	0x03	Link not open
ERSTATOPEN	0x05	Station already open
ERNOCB	0x06	Station control blocks depleted
ERINVINDX	0x07	Invalid link index
ERMAXSTAT	0x08	Limit for number of stations per link reached
ERDIFADDR	0x09	Address different from that on **Request-Open-Station**
ERBADADDR	0x0A	Invalid DLC address

Close(STATION)

Close(STATION) is used by the node to close the STATION LPI connection.

Close(STATION) Request

Flow : NODE ------> DLC

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: CLOSEMSG (0x02)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
clhdr.closqual	CHAR	Close qualifier: REQU (0x01)
clhdr.clstype	CHAR	Close type: STAT (0x11)

Note that the message consists of a buffer header only.

Close(STATION) Response

Flow : DLC ------> NODE

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: CLOSEMSG (0x02)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
clhdr.clsequal	CHAR	Close qualifier - RSPOK (0x02) - RSPERR (0x03)
clhdr.clstype	CHAR	Close type: STAT (0x11)
clhdr.clserr1	INTEGER	Error code

The error codes (for an ERROR-RESPONSE) are defined as:

- 0x03 — Station not open
- 0x04 — Link not connected
- 0x05 — Invalid station index
- 70x06 — Duplicate request

Note
The message consists of a buffer header only.

Note
The Close(STATION) message unconditionally closes the station connection.

Request-Open-Station

Flow : DLC ------> NODE (link connection)

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element if present
numelts	CHAR	Number of buffer elements
msgtype	CHAR	Message type: DLCSTAT (0x11)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
dshdr.dstype	CHAR	Status type: QOPNSTN (0x16)
dshdr.dsqual	CHAR	Station address on XID or mode-set command. Set to 0x01 for 802.2
dshdr.dsmdset	CHAR	Rcv-Set-Mode flag 0x00 XID received 0x01 Mode set command received for example, SNRM for SDLC SABME for 802.2 QSM for X.25

Element field	Type	Description
hdreptr–>eltptr	PTRBFELT	Pointer to next buffer element (NULL is only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+n–1]	Not applicable	XID-starting at first byte of received XID information field. 0x00 NULL XID received (n = 1) and signaling information present.

Optional Signaling Information

Field	Description
dataru[s+n]	Length of data, including this byte
dataru[s+n+1]	Type of data (not used at present)

dataru[s+n+2..s+m]	Address or other identifier data. For example, media access control (MAC) address of remote station X.25 address of remote station

- The signaling information is used by the node to identify the remote station on 802.2 and X.25 links.

Note
If a NULL XID is received and no signaling information is required, the element can be omitted.

Note
If a NULL XID is received and signaling information is required, an 0x00 byte should be put in the element followed by the signaling information.

Note
If the Rcv-Set-Mode flag is set to 0x01, the element can be omitted.

Station-Contacted

Flow : DLC ------> NODE (station connection)

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: DLCSTAT (0x11)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
dshdr.dstype	CHAR	Status type: STNCTCTD (0x17)

Note that this message contains a buffer header only.

Outage

Flow : DLC ------> NODE (link or station connection)

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: DLCSTAT (0x11)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
dshdr.dstype	CHAR	Status type: OUTAGE (0x18)
dshdr.dsqual	CHAR	Outage qualifier
dshdr.dsoutsq	CHAR	Outage subqualifier (optional)

Note the following:

- This message contains a buffer header only.
- Outage qualifier codes are given in Outages.

Status-Resource

Flow : DLC <------> NODE (station connection)

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element: NULL
numelts	CHAR	Number of buffer elements: 0
msgtype	CHAR	Message type: DLCSTAT (0x11)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
dshdr.dstype	CHAR	Status type: RESOURCE (0x04)
dshdr.dlccred	INTEGER	DLC credit

Note

This message contains a buffer header only.

Note

The **dlccred** field indicates that the message sender can receive a further **dlccred DLC-Data** message.

Send-XID

Flow : NODE ------> DLC (link connection)

Header

Field	Type	Description
nxtqptr	PTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PTRBFELT	Pointer to first buffer element
numelts	CHAR	Number of buffer elements
msgtype	CHAR	Message type: DLCSTAT (0x11)
srcl	CHAR	Source locality
srcp	CHAR	Source partner
srci	INTEGER	Source index
destl	CHAR	Destination locality
destp	CHAR	Destination partner
desti	INTEGER	Destination index
dshdr.dstype	CHAR	Status type: SENDXID (0x1A)
dshdr.dsqual	CHAR	Station address on XID

Element

Field	Type	Description
hdreptr–>elteptr	PTRBFELT	Pointer to next buffer element: (NULL is only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>dataru	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+n–1]	Not applicable	XID information frame

Note that the **dshdr.dsqual** field is valid only for primary multipoint connections where station is specified on a multidrop line that the XID should be sent to. In all other cases, it is set to 0xFF.

In situations where the link protocol requires the address field on the XID to be set to a value other than 0xFF (for example, to specify that the XID is a response), it is the responsibility of the link service to set this byte appropriately.

The **Send-XID** message can contain zero elements (**numelts** = 0) or a single, empty element (**hdreptr–>startd** < **hdreptr–>endd**). In these cases, the link service is expected to transmit a NULL XID.

DLC-Data

Flow : DLC <------> NODE

Header

Field	Type	Description
nxtqptr	PPTRBFHDR	Pointer to next buffer header in a queue
hdreptr	PPTRBFELT	Pointer to first buffer element
numelts	CCHAR	Number of buffer elements
msgtype	CCHAR	Message type: DLCDATA (0x10)
srcl	CCHAR	Source locality
srcp	CCHAR	Source partner
srci	INTEGER	Source index
destl	CCHAR	Destination locality
destp	CCHAR	Destination partner
desti	INTEGER	Destination index
ddhdr.ddth01	CCHAR[6]	Transmission header

Element

Field	Type	Description
hdreptr–>eltept r	PPTRBFELT	Pointer to next buffer element: (NULL is only one element)
hdreptr–>startd	INTEGER	Index to start of data in this buffer element's data array: 1
hdreptr–>endd	INTEGER	Index to last byte of data in this buffer element's data array
hdreptr–>datar u	CHAR[SNANBEDA]	Defined as follows, where s = startd–1
dataru[s..s+n–1]	Not applicable	SNA request/response header (RH) if present, and request/response unit (RU) if not prese nt

Configuration Entry Points

The following topics describe the entry points used by SNALink to obtain configuration information.

In This Section

- SNAGetConfigValue
- SNAGetSystemInfo
- pCSInfo

SNAGetConfigValue

SNALink calls the **SNAGetConfigValue** function to obtain the value of a specific configuration parameter.

Syntax

```
USHORT SNAGetConfigValue(
UCHAR *entryName,
VOID *pBuffer,
ULONGbufferLen,
UCHARparmType,
ULONG *pRetLength
);
```

Parameters

entryName

The name of the configuration parameter required.

pBuffer

A pointer to a buffer (if the parameter is a string), or a pointer to a LONGINT (if the parameter is an integer).

bufferLen

The length of the buffer. Only required if the parameter is TYPESTRING.

parmType

TYPESTRING if the parameter is a string.

TYPELONG if the parameter is an integer.

pRetLength

Number of bytes returned if the parameter is TYPESTRING, or number of bytes available if the buffer was too short.

Return Value

NO_ERROR

OK.

ERBADCFG

Error reading configuration file.

ERNOTFND

Entry not found in configuration record.

ERTOOLONG

Data available exceeded the size of the buffer.

ERBADTYPE

A bad type was specified for the *parmType* parameter.

Remarks

It is strongly recommended that SNALink read all required configuration parameters at initialization time (when SNALinkInitialize is called by the Base).

SNAGetSystemInfo

SNALink calls the **SNAGetSystemInfo** function to obtain information about the SNA server and the network operating system.

Syntax

```
INTEGER SNAGetSystemInfo(
struct cs_info *pCSInfo
);
```

Parameters

pCSInfo

Pointer to a buffer supplied by the application that contains a data structure in which system information is returned. The application must set the **length** field in this data structure. (For details, see Remarks.) The other fields should be set to nulls or blanks.

Return Value

NO_ERROR

OK.

ERNOCFGSVR

No configuration file server available.

ERMOREDATA

Supplied buffer was too small.

Remarks

The application must set the **length** parameter to the length of the **cs_info** structure (86 bytes in the current version). Any other value is rejected. This parameter is used to ensure compatibility with future versions. An application supplying this length will always obtain the information shown, but in future versions it may be possible to specify larger values and obtain further information.

On successful return, the **cs_info** data structure contains the version number of the SNA server, the path to the current configuration file, and the network operating system over which the SNA server is running.

If there is no configuration file server available, only the version number fields are valid. The other fields should not be checked.

pCSInfo

Syntax

```
struct cs_info {
    unsigned short  length;
    unsigned char   major_ver;
    unsigned char   minor_ver;
    unsigned char   config_share[80];
    unsigned short  nos;
};
```

Members

length

Length of the data structure supplied by the application.

major_ver

Major version number:

1 for Communications Server 1.1

2 for SNA Server 2.0 or 2.1

3 for SNA Server 3.0

4 for SNA Server 4.0

minor_ver

Minor version number (decimal):

10 for Communications Server 1.1

00 for SNA Server 2.0

20 for SNA Server 2.1

00 for SNA Server 3.0

00 for SNA Server 4.0

config_share[80]

The name of the share point of the current configuration file (\\server\share\, for example). This path name must be a null-terminated string.

nos

Transport protocol in use:

bit 0: LAN Manager/LAN Server (named pipes)

bit 1: NetWare (IPX/SPX)

bit 2: AppleTalk (Not supported in Host Integration Server 2009)

bit 3: Banyan VINES (VINES IP)

bit 4: TCP/IP

Setup Functions

This section provides a reference for the functions used with the integrated link service dynamic-link library (DLL) architecture as well as INF-based Setup.

This section contains:

- Integrated Link Service Configuration Functions
- Inf-Based Setup Functions

Integrated Link Service Configuration Functions

This section provides a reference for exported dynamic-link library (DLL) entry points and utility functions used when building an integrated link service configuration dynamic-link library (DLL).

This section contains:

- Functions Exported from a Link Service Configuration DLL
- Utility Functions Used by a Link Service Configuration DLL

Functions Exported from a Link Service Configuration DLL

This section provides a reference for functions that must be exported from an integrated link service configuration dynamic-link library (DLL).

In This Section

- CommandLineAdd
- ConfigureLinkService
- ConfigureLinkServiceEx
- DisplayHelpInfo
- RemoveAllLinkServices
- RemoveLinkService

CommandLineAdd

The **CommandLineAdd** function is used to add a new link service using a command-line interface. This function must be exported from a link service configuration dynamic-link library (DLL) supplied with each link service.

Syntax

```
__declspec(dllexport) BOOL WINAPI CommandLineAdd(  LPSTRszCommandLine,
  LPSTR *szConfigInfo,
  LPDWORD dConfigInfoSize);
```

Parameters

szCommandLine

This supplied parameter specifies the command line containing information on the computer and link service to be configured.

szConfigInfo

This supplied and returned parameter points to a configuration buffer that is used to configure the link service.

dConfigInfoSize

This supplied parameter specifies the size of the sz*ConfigInfo* configuration buffer .

Return Value

TRUE

The function executed successfully.

FALSE

One or more of the parameters passed to this function are invalid or the function failed.

ConfigureLinkService

The **ConfigureLinkService** function is used to add or modify a link service. This function must be exported from a link service configuration dynamic-link library (DLL) supplied with each link service.

Syntax

```
          __declspec(dllexport) BOOL WINAPI ConfigureLinkService(
    LPSTR szComputerName,
    LPSTRszLinkServiceTitle);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer that is to be configured.

szLinkServiceTitle

This supplied parameter specifies the title of the link service that is to be configured.

Return Value

TRUE

The function executed successfully and network bindings need to be recalculated.

FALSE

One or more of the parameters passed to this function are invalid or network bindings do not need to be recalculated.

ConfigureLinkServiceEx

The **ConfigureLinkServiceEx** function is used to add or modify a link service. This function must be exported from a link service configuration dynamic-link library (DLL) supplied with each link service.

Syntax

```
          __declspec(dllexport) BOOL WINAPI ConfigureLinkServiceEx(
    LPSTRszComputerName,
    LPSTRszLinkServiceTitle,
    LPSTR* pvConfigInfo,
    LPDWORD dConfigInfoSize);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer that is to be configured.

szLinkServiceTitle

This supplied parameter specifies the title of the link service that is to be configured.

pvConfigInfo

This supplied and returned parameter points to a configuration buffer that is used to configure the link service.

dConfigInfoSize

This supplied parameter specifies the size of the *pvConfigInfo* configuration buffer.

Return Value

TRUE

The function executed successfully and network bindings need to be recalculated.

FALSE

One or more of the parameters passed to this function are invalid or network bindings do not need to be recalculated.

DisplayHelpInfo

The **DisplayHelpInfo** function is used to generate help information used by the command-line interface to a link service dynamic-link library (DLL). This function must be exported from a link service configuration DLL supplied with each link service.

Syntax

```
            __declspec(dllexport) BOOL WINAPI DisplayHelpInfo(
    LPSTR*szHelpInfoBuffer);
```

Parameters

szHelpInfoBuffer

This supplied and returned parameter points to a buffer that on successful return contains help information that can be used to configure the link service.

Return Value

TRUE

The function executed successfully.

FALSE

The parameter passed to this function is invalid or the function failed.

RemoveAllLinkServices

The **RemoveAllLinkServices** function is used to remove all link services from a machine. This function must be exported from a link service configuration dynamic-link library (DLL) supplied with each link service.

Syntax

```
          __declspec(dllexport) BOOL WINAPI RemoveAllLinkServices(
   LPSTRszComputerName);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer that is to have all link services removed.

Return Value

TRUE

The function executed successfully and network bindings need to be recalculated.

FALSE

The parameter passed to this function is invalid or network bindings do not need to be recalculated.

RemoveLinkService

The **RemoveLinkService** function is used to remove a link service. This function must be exported from a link service configuration dynamic-link library (DLL) supplied with each link service.

Syntax

```
          __declspec(dllexport) BOOL WINAPI RemoveLinkService(
    LPSTRszComputerName,
    LPSTRszLinkServiceTitle);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer that is to have the link service removed.

szLinkServiceTitle

This supplied parameter specifies the title of the link service that is to be removed.

Return Value

TRUE

The function executed successfully and network bindings need to be recalculated.

FALSE

One or more of the parameters passed to this function are invalid or network bindings do not need to be recalculated.

This section contains:

- Utility Functions Used by a Link Service Configuration DLL

Utility Functions Used by a Link Service Configuration DLL

This section provides a reference for utility functions used by an integrated link service configuration dynamic-link library (DLL).

This section contains:

AddPerfmonCounters

The **AddPerfmonCounters** function is used to add Perfmon counters to a link service. This utility function is used to construct an integrated link service configuration dynamic-link library (DLL).

Syntax

```
          void AddPerfmonCounters(
    LPSTRpszComputerName,
    LPSTRpszService);
```

Parameters

pszComputerName

This supplied parameter specifies the name of the computer that is to have Perfmon counters added.

pszService

This supplied parameter specifies the name of the link service that is to have Perfmon counters added.

Return Value

None.

Remarks

SNA RPC Service must be running or an error message will indicate a failure.

bCreateService

The **bCreateService** function is used to create a service on a computer for a link service. This utility function is used to construct an integrated link service configuration dynamic-link library (DLL).

Syntax

```
          BOOL bCreateService(
    LPSTRszComputerName,
    LPSTRszServiceName,
    LPSTRszServicePath,
    LPSTRszServiceDependencies,
    DWORDdServiceType,
    DWORDdServiceLoadType,
    LPSTRszDomainName,
    LPSTRszUserid,
    LPSTRszPassword);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer to create the service on.

szServiceName

This supplied parameter specifies the name of the link service that is to be created. This parameter is passed unchanged to the Microsoft® Windows® 2000 **CreateService** function.

szServicePath

This supplied parameter specifies the binary path to the link service that is to be created. This parameter is passed unchanged to the Windows 2000 **CreateService** function.

szServiceDependencies

This supplied parameter specifies the service dependencies of the link service that is to be created. This parameter is passed unchanged to the Windows 2000 **CreateService** function.

dServiceType

This supplied parameter specifies the type of service that is to be created. This parameter is passed unchanged to the Windows 2000 **CreateService** function.

dServiceLoadType

This supplied parameter specifies the load type of service that is to be created. This parameter is passed unchanged to the Windows 2000 **CreateService** function.

szDomainName

This supplied parameter specifies the domain name for the service to run in.

szUserid

This supplied parameter specifies the user identifier for the service to run in.

szPassword

This supplied parameter specifies the password for the domain account.

Return Value

TRUE

The function executed successfully and the service was created.

FALSE

One or more of the parameters passed to this function are invalid or the function failed.

Remarks

If the *szUserid* parameter is not supplied, then the *szDomainName* parameter is used to construct the Account parameter passed to the Windows 2000 **CreateService** function.

bDeleteService

The **bDeleteService** function is used to delete a service on a computer for a link service. This utility function is used to construct an integrated link service configuration dynamic-link library (DLL).

Syntax

```
          BOOL bDeleteService(
    LPSTRszComputerName,
    LPSTRszServiceName);
```

Parameters

szComputerName

This supplied parameter specifies the name of the computer to delete the service on.

szServiceName

This supplied parameter specifies the name of the service that is to be deleted. This parameter is passed unchanged to the Microsoft® Windows® 2000 **OpenService** function.

Return Value

TRUE

The function executed successfully and the service was deleted.

FALSE

One or more of the parameters passed to this function are invalid or the function failed.

bStopService

The **bStopService** function is used to stop a service running on a computer for a link service. This utility function is used to construct an integrated link service configuration dynamic-link library (DLL).

Syntax

```
          BOOL bStopService(
    LPSTRszServiceName,
    LPSTRszComputerName);
```

Parameters

szServiceName

This supplied parameter specifies the name of the service that is to be stopped. This parameter is passed unchanged to the Microsoft® Windows® 2000 **OpenService** function.

szComputerName

This supplied parameter specifies the name of the computer to stop the service on.

Return Value

TRUE

The function executed successfully and the service was stopped.

FALSE

One or more of the parameters passed to this function are invalid or the function failed.

CheckForExistingLinkService

The **CheckForExistingLinkService** function is used to check to see if a link service of this type exists with this title. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL bCreateService(
    HKEY *hGlobalKey,
    LPSTRszLinkRegistryRoot,
    LPSTRszTitle);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry to search.

szLinkRegistryRoot

This supplied parameter specifies the registry root for this type of link service to search for.

szTitle

This supplied parameter specifies the title of the link service to search for.

Return Value

TRUE

The link service was located.

FALSE

One or more of the parameters passed to this function are invalid or the link service was not located.

ConvertHexStringToDWORD

The **ConvertHexStringToDWORD** function is used to convert a hexadecimal string to a DWORD value. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL ConvertHexStringToDWORD(
    LPSTRszHexString,
    LPDWORDdHexValue);
```

Parameters

szHexString

This supplied parameter specifies the hexadecimal string to be converted.

dHexValue

This supplied and returned parameter contains the DWORD value of the hexadecimal string if the function was successful.

Return Value

TRUE

The function executed successfully and the hexadecimal string was converted.

FALSE

One or more of the parameters passed to this function are invalid or the function failed.

Remarks

This function scans until a nonhexadecimal character is encountered. The hexadecimal formats recognized are xnnnn, 0xnnnn, or nnnn.

ExtractNextParameter

The **ExtractNextParameter** function is used to get the next parameter from a buffer. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL ExtractNextParameter(
    LPSTRszSourceBuffer,
    LPSTRszParameter,
    LPDWORDdStartIndex);
```

Parameters

szSourceBuffer

This supplied parameter specifies the source buffer.

szParameter

This supplied and returned parameter specifies the return buffer for parameters.

dStartIndex

This supplied parameter contains the DWORD index to begin parameter scan.

Return Value

TRUE

The function executed successfully and the next parameter was located and returned in the *szParameter* argument.

FALSE

The function failed.

Remarks

Parameters are delimited by a space character. Quotes can be used to include spaces in a parameter.

fAddRegistryEntry

The **fAddRegistryEntry** function is used to add a new registry value to the registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL fAddRegistryEntry(
    HKEY *hGlobalKey,
    char *szRegistryValue,
    char *szRegistryData,
    DWORDdType,
    DWORDdSize);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry to modify.

szRegistryValue

This supplied parameter specifies the registry value name to add.

szRegistryData

This supplied parameter specifies the registry value data to add.

dType

This supplied parameter specifies the registry value type. This parameter is supplied unchanged to the Win32® **RegSetValueEx** function.

dSize

This supplied parameter specifies the size of the registry value data. This parameter is supplied unchanged to the Win32 **RegSetValueEx** function.

Return Value

TRUE

The function executed successfully and the registry entry was added.

FALSE

The function failed and the registry entry was not added.

fCanWeAdministerRemoteBox

The **fCanWeAdministerRemoteBox** function is used to determine if the caller has administrative privileges on the remote computer. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
            BOOL fCanWeAdministerRemoteBox(
    HKEY *hGlobalKey);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle to the remote computer's registry.

Return Value

TRUE

The caller has administrative privileges on the remote computer.

FALSE

The caller lacks administrative privileges.

fConnectRegistry

The **fConnectRegistry** function is used to connect to a remote computer's registry and return a handle to the remote registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL fConnectRegistry(
    HKEY *hGlobalKey,
    LPSTR *szComputerName);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry to connect to.

szComputerName

This supplied parameter specifies the computer name to connect to.

Return Value

TRUE

The function executed successfully and the function was able to connect to the registry.

FALSE

The function failed.

fDisconnectRegistry

The **fDisconnectRegistry** function is used to disconnect from a remote computer's registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
            BOOL fDisconnectRegistry(
    HKEY *hGlobalKey);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry from which to disconnect.

Return Value

TRUE

The function executed successfully and the function was able to disconnect from the registry.

FALSE

The function failed.

fFindAndReplaceString

The **fFindAndReplaceString** function is used to find and replace a substring within a string. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL fFindAndReplaceString(
    LPSTRszStringBuffer,
    LPSTRszSearchString,
    LPSTRszReplaceString);
```

Parameters

szStringBuffer

This supplied parameter specifies the string buffer to search.

szSearchString

This supplied parameter specifies the string to search for.

szReplaceString

This supplied parameter specifies the replacement string.

Return Value

TRUE

The string was found.

FALSE

The string was not found.

fFindString

The **fFindString** function is used to determine if a string exists within a string buffer. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          BOOL fFindString(
    LPSTRszStringBuffer,
    LPSTRszSearchString);
```

Parameters

szStringBuffer

This supplied parameter specifies the string buffer to search.

szSearchString

This supplied parameter specifies the string to search for.

Return Value

TRUE

The string was found.

FALSE

The string was not found.

fFindStringInMultiSZ

The **fFindStringInMultiSZ** function is used to determine if string exists in a REG_MULTI_SZ string list and return entire string. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
        BOOL fFindString(
        LPSTR
        szStringBuffer,
    LPSTRszSearchString,
    LPSTRszFoundString);
```

Parameters

szStringBuffer

This supplied parameter specifies the string buffer to search.

szSearchString

This supplied parameter specifies the string to search for.

szFoundString

This supplied and returned parameter specifies the full string containing string if successful.

Return Value

TRUE

The string was found and returned.

FALSE

The string was not found.

fQueryRegistryValue

The **fQueryRegistryValue** function is used to query a value from the registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
        BOOL fQueryRegistryValue(
        HKEY *
        hGlobalKey,
    LPSTRszRegistryKey,
    LPSTRszRegistryValue,
    LPSTRszRegistryData,
    LPDWORDdSize);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry.

szRegistryKey

This supplied parameter specifies the registry key.

szRegistryValue

This supplied parameter specifies the registry value name.

szRegistryData

This supplied parameter specifies the registry value data.

dSize

This supplied parameter specifies the size of the registry data.

Return Value

TRUE

The function executed successfully and the function was able to connect to the registry.

FALSE

The function failed.

fRegistryKeyExists

The **fRegistryKeyExists** function is used to determine if a registry key already exists in the registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
            BOOL fRegistryKeyExists (
   HKEY *hGlobalKey,
   LPSTRszRegistryKey
 );
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry.

szRegistryKey

This supplied parameter specifies the registry key.

Return Value

TRUE

The registry key exists.

FALSE

The function failed or the registry key does not exist.

fRemoveRegistryEntry

The **fRemoveRegistryEntry** function is used to remove a registry key from the registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
            BOOL
            fRemoveRegistryEntry
            (
            HKEY *
            hGlobalKey,
    char *szRegistryKey
);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry.

szRegistryKey

This supplied parameter specifies the registry key.

Return Value

TRUE

The function was successful and the registry key was removed.

FALSE

The function failed or the registry key could not be removed.

fRemoveRegistryValue

The **fRemoveRegistryValue** function is used to remove a registry value from the registry. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
            BOOL fRemoveRegistryValue (
            HKEY *
            hGlobalKey,
    char *szRegistryKey,
    char *szRegistryValue
);
```

Parameters

hGlobalKey

This supplied parameter specifies the handle of the registry.

szRegistryKey

This supplied parameter specifies the registry key.

szRegistryValue

This supplied parameter specifies the registry value to remove.

Return Value

TRUE

The function was successful and the registry value was removed.

FALSE

The function failed or the registry value could not be removed.

fStringCompare

The **fStringCompare** function is used to determine if string exists in another string. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
BOOL fStringCompare (
  LPSTR lpszString1,
  LPSTR lpszString2
);
```

Parameters

lpszString1

This supplied parameter specifies the string to search for.

lpszString2

This supplied parameter specifies the string to compare.

Return Value

TRUE

The string was found.

FALSE

The string was not found.

LoadStringResource

The **LoadStringResource** function is used to load a string from a string resource. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          void LoadStringResource (
   DWORDdStringResource,
   LPSTRpszString);
```

Parameters

dStringResource

This supplied parameter specifies the resource ID of the string resource.

pszString

This supplied and returned parameter specifies the buffer to place the string in.

Return Value

None

ParseNextField

The **ParseNextField** function is used to parse and return the next field from string. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
        void ParseNextField(
        LPSTR
        szParseString,
    LPSTRszField,
    charscDelimiter,
    LPDWORDdStartIndex);
```

Parameters

szParseString

This supplied parameter specifies the string to parse.

szField

This supplied and returned specifies the return buffer for the next field.

scDelimiter

This supplied parameter specifies the delimiter character for the end of a field.

dStartIndex

This supplied parameter specifies a pointer to the index in bytes from beginning of the *szParseString* to start the search from.

Return Value

TRUE

The next field was found.

FALSE

The next field was not found.

Note
The **'^'** character can be used as an escape character to allow the delimiter to be used.

RemovePerfmonCounters

The **RemovePerfmonCounters** function is used to remove counters from a link service. This utility function is used to construct an integrated link service configuration DLL.

Syntax

```
          void RemovePerfmonCounters(
    LPSTRpszComputerName,
    LPSTRpszService);
```

Parameters

pszComputerName

This supplied parameter specifies the name of the computer that is to have Perfmon counters removed.

pszService

This supplied parameter specifies the name of the link service that is to have Perfmon counters removed.

Return Value

None.

Remarks

SNA RPC Service must be running or an error message will indicate a failure.

Inf-Based Setup Functions

This section provides a reference for some of the useful entry points in the .inf file that contains utility functions. The file name is in the variable *UtilityInf*; usually set to SNAUTILS.INF.

This section contains:

CreateSNARegEntry

The **CreateSNARegEntry** function creates the necessary entries for an instance in the **SOFTWARE\Microsoft** registry tree. If the product is not already in the registry, it creates an entry for the product. It then creates an entry for the particular instance of the product and for the **NetRules** key under that entry. This function leaves open handles to all the important subkeys for further use.

Parameters
Argument 0

Name of the top-level registry node to use. This should be a full registry path. For most scenarios, this is the value held in the *ProductRegBase* variable (**SOFTWARE\Microsoft**).

Argument 1

Name of the product. This is the name of the key that will be created for this product.

Argument 2

Instance index. This is the index of this particular instance of this product.

Return Values
Return 0

Status of the operation:

- STATUS_SUCCESSFUL: Operation succeeded.
- STATUS_FAILED: Operation failed.

Return 1

Handle to the top-level registry node.

Return 2

Handle to the products registry key under the top-level node.

Return 3

Handle to the instance entry under the product key.

Return 4

Handle to the NetRules entry under the instance key.

CreateSNAService

The **CreateSNAService** function creates the necessary entries for an instance in the **Services** registry tree. This function adds particular values that are necessary for the service as well as all the subkeys under the service key, including **Parameters** and **ExtraParameters**.

Parameters
Argument 0

Name of the service to be created.

Argument 1

Type of SNA Service (*SNAServiceType* variable).

Argument 2

Image path of this component (*ImagePath* variable).

Argument 3

Dependency list (*ProductDepends* variable).

Argument 4

Parameter list (*ProductParams* variable).

Argument 5

Extra parameter list (*ProductExtraParams* variable).

Argument 6

Event Log message file.

Argument 7

Event types supported (usually 0x07).

Return Values
Return 0

Status of the operation:

- STATUS_SUCCESSFUL: Operation succeeded.
- STATUS_FAILED: Operation failed.

Return 1

Handle to the service key.

Return 2

Handle to the **Parameters** key under the service key.

Return 3

Handle to the **ExtraParameters** key under the **Parameters** key.

DeleteSNAService

The **DeleteSNAService** function deletes a particular service using the Service Control Manager. All the keys for the service are deleted as well.

Parameters
Argument 0

Name of the service to be deleted.

Return Values
Return 0

Status of the operation.

EnterServiceName

The **EnterServiceName** function presents the user with an algorithmically determined service name for a component and allows the user to change it before returning the final value. This function ensures that the new service name is unique in the Service Control Architecture before accepting it.

Parameters
Argument 0

Title of the product the user should be queried about.

Argument 1

Default service name prefix.

Argument 2

Index for this instance of the product. The algorithm uses this index to determine the default name.

Return Values
Return 0

Status of the operation:

- STATUS_SUCCESSFUL: Operation succeeded.
- STATUS_NOSUCHLANGUAGE: The language specified is not supported.
- STATUS_USERCANCEL: User pressed the **Cancel** button.

Return 1

Service name that the user entered.

FindNextAvailableIndex

The **FindNextAvailableIndex** function is used to determine the index a new instance should receive. For example, if the current indexes in use are { 01, 02, 04 }, this function would return 03 as its return value.

Parameters
Argument 0

A list of the indexes currently in use. This list can be obtained by using the FindSNAProductServices function.

Return Values
Return 0

Status of the operation:

- STATUS_SUCCESSFUL: Operation succeeded.
- STATUS_FAILED: Operation failed.

Return 1

First available index for the list.

FindSNAProductServices

The **FindSNAProductServices** function enumerates all instances of a product. It returns lists of information for the instances that can be used in the script. This function can also be used to determine whether or not a product exists in the registry by analyzing the return status.

Parameters
Argument 0

Name of top-level registry node to use.

Argument 1

Name of the product.

Return Values
Return 0

Status of the operation:

- STATUS_SUCCESSFUL: Operation succeeded.
- STATUS_NOSUCHPRODUCT: The product does not exist in the registry.
- STATUS_FAILED: Operation failed.

Return 1

List of indexes for the instances of this product.

Return 2

List of service names for the instances of this product.

Return 3

List of titles for the instances of this product.

Return 4

List of descriptions for the instances of this product.

FindSNARegEntry

The **FindSNARegEntry** function is written as a parallel to CreateSNARegEntry. When called, it attempts to open all of the necessary registry keys and return open handles to them.

Parameters
Argument 0

Name of the top-level registry node to use.

Argument 1

Name of the product.

Argument 2

Instance index.

Return Values
Return 0

Status of the operation.

Return 1

Handle to the top-level registry node.

Return 2

Handle to the products registry key under the top-level node.

Return 3

Handle to the instance entry under the product key.

Return 4

Handle to the NetRules entry under the instance key.

FindSNAService

The **FindSNAService** function is written as a parallel to CreateSNAService. It is written to provide an easy way to access the keys for a particular service.

Parameters
Argument 0

Name of the service to look for.

Return Values
Return 0

Status of the operation.

Return 1

Handle to the service key.

Return 2

Handle to the **Parameters** key under the service key.

Return 3

Handle to the **ExtraParameters** key under the **Parameters** key.

GrepUniqueServiceInfo

The **GrepUniqueServiceInfo** function is used to determine information about a particular instance when only one of the four elements (index, name, title, or description) is available. Because there is no way to determine the position of an element in a list, it is hard to figure out the respective name, title, or description of an instance given only the index. This function searches the list and returns the rest of the information about the instance.

Parameters
Argument 0

Type of information to search:

- INDEX: Search the list of indexes.
- NAME: Search the list of service names.
- TITLE: Search the list of titles.
- DESC: Search the list of descriptions.

Argument 1

Keyword to search for in the list.

Argument 2

List of indexes for the instances of this product.

Argument 3

List of service names for the instances of this product.

Argument 4

List of titles for the instances of this product.

Argument 5

List of descriptions for the instances of this product.

Return Values
Return 0

Status of the operation.

Return 1

Index for this instance of the product.

Return 2

Service name for this instance of the product.

Return 3

Title for this instance of the product.

Return 4

Description for this instance of the product.

SetupMessage

The **SetupMessage** function displays a dialog box with user-defined text plus **OK** and **Cancel** buttons. It is useful for debugging.

Parameters
Argument 0

Language to use (*STF_LANGUAGE*).

Argument 1

Which icon to display in the dialog box: STATUS, WARNING, NONFATAL, and so on.

Argument 2

The text to be displayed. Can contain line feeds using the LF symbol.

Return Values
Return 0

Status of the operation.

IOCTL Commands

This section provides reference information about the IOCTL functions.

This section contains:

Function 0x41: Set Event/Semaphore Handle

This function supplies the driver with the handle of an event that can be used for signaling the SNALink software.

Parameters
IRP.UserEvent

This parameter is taken from the IOCTL call, and is an event handle. The handle must refer to an Event/Semaphore owned by the SNALink process. The driver sets the event to indicate the completion of a transmission or the availability of received data or status. Although not required by the driver, the event passed here by the SNALink is normally the global Base event.

Return Values

If the supplied parameter is NULL, the function returns a status of STATUS_INVALID_PARAMETER, with additional information of INFO_SET_EVENT_NO_EVENT. For any other illegal parameter, an exception is raised.

Remarks

This function should be called only once, immediately after the **OPEN** is issued. The event is set when:

- IFrames are transmitted from the driver buffers.
- IFrames are received into the driver buffers.
- IStatus information is updated in the Interface Record (see function 0x64).

Function 0x42: Set Link Characteristics

This function sets the link protocol parameters required by the driver.

Packet Formats

Type	Description
WORD	Frame Size
DWORD	Data rate
BYTE	Station address 1
BYTE	Station address 2
BYTE	Link Options
BYTE	Reserved

Parameters

Frame Size(packet format WORD)

Indicates to the driver the minimum amount of contiguous receiver buffering that must be available for any individual frame.

Data Rate(packet format DWORD)

Line speed in bits per second. If *Link Options* bit 3 is not set, this field is ignored.

Station Address 1(packet format BYTE)

Station Address 2(packet format BYTE)

Station addresses that the user wishes to receive on if selective reception is to be used (typically for multidropped secondaries). Only frames of data beginning with either of these values will be passed to the user—others are ignored or discarded.

If the SNALink wishes to listen on only one station address, *Station Address 2* should be set to zero.

A value of zero in both fields indicates that all error-free received frames are to be passed to the SNALink, regardless of the contents of their first address byte.

Link Options(packet format BYTE)

Link Options is a bitmap. The default is all values set to zero. The bits are used as shown in the following table. Note that not all of these options are supported by the standard Microsoft Host Integration Server 2009 synchronous card drivers.

Bit	Value
Bit 7	1 = Four wire (constant RTS/CTS). 0 = Two wire (switched RTS/CTS).
Bit 6	1 = NRZI encoding. 0 = NRZ encoding.
Bit 5	1 = HDLC level 1 conventions. 0 = SDLC level 1 conventions.
Bit 4	1 = Full-duplex (simultaneous 2-way) data. 0 = Half-duplex (alternating 1-way) data.
Bit 3	1 = Generate internal clocking. 0 = Take external clocking.
Bit 2	1 = Use DMA if available. 0 = Do not use DMA on this link.
Bit 1	1 = Reset all statistics to zero. 0 = Leave accumulated statistics as is.

Bit 0	Reserved.

Reserved(packet format BYTE).

Reserved.

Note
Not all of the above options are supported by the standard Host Integration Server synchronous card drivers.

Return Values

IoStatus.Status	**IoStatus.Information**	**Description**
STATUS_INVALID_PARAMETER	IO_ERR_LINKCHARBUF_ WRONG_SIZE	
STATUS_INVALID_PARAMETER	IO_ERR_FRAME_BUF_ TOO_SMALL	Buffer must be at least 268 bytes.
STATUS_INVALID_PARAMETER	IO_ERR_FRAME_BUF_ TOO_BIG	Buffer maximum size is 2048 bytes.
STATUS_INVALID_PARAMETER	IO_ERR_NO_CLOCKS	No internal clocking available.
STATUS_DATA_ERROR	IO_ERR_HARDWARE_ 8273CMD_TIMEOUT	
STATUS_SUCCESS	IO_ERR_NO_DMA_FDX	DMA requested, but cannot be used.

Remarks

The driver should always start the receiver after processing this request. If either the transmitter or receiver is active when this request is issued, the driver stops the current frame before resetting the link characteristics, and then restarts the previous operation.

Link Service DLLs that support the synchronous dumb card interface use the following registry entries to the control this feature.

SYSTEM\CurrentControlSet\Services\<linkService>\Parameters

where **<linkService>** is the name of the link service.

Under this node, the following entries and values must be entered or modified:

A node called **ExtraParameters** must be created or modified with the following registry entries and values:

InternalClock

The value of this entry should be defined and set to a REG_DWORD of any value to enable the internal clock.

InternalClockRate

The value of this entry should be set to a REG_DWORD equal to the number of bits per second.

Function 0x43: Set V24 Output Status

This function allows the SNALink software to alter the modem output status on the adapter V.24 interface. There is no parameter or data packet on this request. The relevant V.24 settings are put into the driver interface record (see function 0x61) by the SNALink prior to calling the driver.

Return Values

IoStatus.Status	**IoStatus.Information**
STATUS_DATA_ERROR	IO_ERR_HARDWARE_8273CMD_TIMEOUT

Function 0x44: Transmit Frame

The SNALink calls this function to transfer a frame of data to the driver.

Parameters

IRP.CurrentStackLocation.OutputBufferLength

Frame length—the size of the frame pointed to by the data buffer pointer. The frame includes the control, address, and information field (if present), but no allowance should be made for flags or CRC bytes.

IRP.UserBuffer

Frame data—the contents of the frame.

Return Values

IoStatus.Status	**IoStatus.Information**
STATUS_BUFFER_TOO_SMALL	IO_ERR_TX_BUFFER_FULL
STATUS_INVALID_PARAMETER	IO_ERR_TX_FRAME_TOO_BIG

Refer also to the description of the interface record—the driver updates a field within the interface record reflecting the amount of buffer space available.

Remarks

In two-wire configurations, the driver must raise RTS and wait for CTS from the modem before initiating a transmission. The driver should then drop RTS when all frames have been transmitted. The driver assumes that the transmission is complete when both of the following are true:

- The transmit buffer becomes empty (if the link is configured as half-duplex).
- The last frame transmitted had the Poll/Final bit set in the second byte.

Function 0x45: Abort Transmitter

The SNALink calls this function to clear down the driver's transmitter.

Remarks

This request causes the driver to stop the current frame transmission and to flush its internal buffers of any further data pending transmission. In two-wire configurations, the driver should also drop RTS.

Function 0x46: Abort Receiver

The SNALink calls this function to clear down the driver's receiver.

Remarks

This request causes the driver to stop the receiver and to flush its internal buffers of any received data. It should be used to clear down the receiver, for instance, before altering the link characteristics.

Function 0x47: Off-Board Load

This function is not supported.

Function 0x61: Get/Set Interface Record

This function has been superseded by Function 0x64: Read Interface Record.

Function 0x62: Get V24 Status

The SNALink calls this function to read the current state of the modem interface lines. No parameter or data packet is passed. This request causes the driver to update the Input V.24 Status field in the driver interface record.

Function 0x63: Receive Frame

The SNALink calls this function to read a data frame from the driver's buffers.

Parameters

IRP.CurrentStackLocation.OutputBufferLength

Frame length—the size of the frame transferred into the data buffer by the driver. The frame includes the control, address, and information field (if present), but no flags or CRC bytes. When the request is issued, frame length is set to the maximum length of the buffer addressed by the data packet pointer.

IRP.UserBuffer

Frame data—the contents of the frame that has been received.

Return Values

IoStatus.Status	**IoStatus.Information**
STATUS_BUFFER_TOO_SMALL	None

Remarks

The driver transfers the next available received frame to the supplied buffer. Note that if the buffer is not large enough, a buffer overflow error is returned. This allows the application to decide if oversize frames are an error. If not, then a second attempt to read should be made, using a buffer at least Frame Length bytes long. A length of zero is returned if there are no frames ready to be received.

Function 0x64: Read Interface Record

This function reads the driver's interface record and copies it into the buffer passed by the SNALink. The buffer must be allocated by the SNALink prior to making this call.

Parameters
IRP.System.Buffer

Interface Record Address (IN)—a 32-bit pointer to the driver's interface record area.

The interface record format is as follows:

Description	Type
Received Frames	**int**
Transmit Buffer Space	**int**
Status Events	**int**
Input V.24 Status	**UCHAR**
Output V.24 Status	**UCHAR**
Statistics Counters	**int[11]**

- Received Frames is the number of frames currently queued in the driver receive buffers.
- Transmit Buffer Space is used to signal to the SNALink:

  Whether more frames can be provided to the driver.

  Whether the driver still has frames waiting to be sent.

  The Transmit Buffer Space field indicates the maximum frame size that the driver can currently accept. This is updated after each successful Transmit Frame IOCTL, and should be checked by the SNALink before sending further frames to the driver.
- Status Events is a count of the total number of increments made to the Statistics Counters.
- Input V.24 Status is a bitmap of the current logical state of the input interface lines, a value of 1 meaning ON and a value of 0 meaning OFF. The pins are mapped as follows:

Bit number	V.24 circuit name	Circuit number	RS-232C pin
7 - 5	Reserved	142	25
4	Test Indicator	125	22
3	Calling Indicator	125	22
2	RLSD (commonly DCD)	109	8
1	Data Set Ready (DSR)	107	6
0	Clear to send (CTS)	106	5

- Output V.24 Status is a bitmap of the current logical state of the output interface lines, a value of 1 meaning ON and a

value of 0 meaning OFF. The pins are mapped as follows:

Bit number	V.24 circuit name	Circuit number	RS-232C pin
7 - 5	Reserved		
4	Maintenance Test	140	18
3	Select Standby	116	11
2	Data Signal Rate Selector	111	23
1	Data Terminal Ready (DTR)	108/2	22
0	Request to Send (RTS)	105	4

Note that the driver will raise and lower RTS as necessary while transmitting, reflecting the state of RTS in the output V.24 status field. The application should not, therefore, try to manipulate RTS.

- The Statistics Counters are running counts of various kinds of link status information.The events they relate to are:

Counter number	Description
1	Frames received with incorrect CRC.
2	Frames larger than the maximum size received.
3	Frames smaller than 32 bits received.
4	Frames received that are not a multiple of 8 bits.
5	Aborted frames received.
6	Transmitter underruns.
7	Receiver overruns.
8	RLSD drops in mid-reception.
9	CTS drops in mid-transmission.
10	DSR drops.
11	Hardware failures (adapter or modem).

Remarks

The interface record provides a fast mechanism to use to decide whether a frame can be transmitted, whether there are any frames to be received, and so on. The driver maintains this information. Each time the SNALink requires this information, it calls **Read Interface Record** to get a copy of the current interface record. After each call, the driver clears the statistics counters in its own interface record. The V.24 status and transmit and receive data information are unchanged.

SNA Modem API

This section provides reference material for the SNA Modem API structure and functions.

In This Section

- MODEM_STATUS
- SNAModemInitialize
- SNAModemAddLink
- SNAModemDeleteLink
- SNAModemTerminate

MODEM_STATUS

A **MODEM_STATUS** structure for each SNA link contains status information used by the SNA Modem Status interface.

Syntax

```
struct _ModemStatus{
....char LSName[12];
    char V24In;
    char V24Out;
    unsigned short RxFrameCount;
    unsigned short TxFrameCount;
    char Reserved[6];
} MODEM_STATUS;
```

Remarks

Members

LSName

A character array containing the link service name.

V24In

A set of bit fields representing the V.24 input flags that determine which signal lines to mask. These bit fields can be joined with **OR** to create a complete mask. The defined bit fields for **V24In** are as follows:

MASK_CTS Mask the clear to send line.

MASK_DSR Mask the data set ready line.

MASK_DCD Mask the data carrier detect line.

MASK_DRI Mask the data ring indicator line.

V24Out

A set of bit fields representing the V.24 output flags that determine which signal lines to mask. These bit fields can be joined with **OR** to create a complete mask. The defined bit fields for **V24Out** are as follows:

MASK_RTS Mask the request to send line.

MASK_DTR Mask the data terminal ready line.

RxFrameCount

A count of received frames.

TxFrameCount

A count of transmitted frames.

Reserved

Padding for future expansion.

SNAModemInitialize

The **SNAModemInitialize** function should be called once per link service process at initialization. This function initializes the communication path to the SNA Modem application. The ideal place to call this function is in the **SNALinkInitialize** function.

Syntax

```
void SNAModemInitialize();
```

See Also

Reference

SNALinkInitialize

SNAModemAddLink

The **SNAModemAddLink** function should be called once per link initialization. For link services that support more than a single SNA link, this call can be made multiple times. For link services that support only a single link, this call can be made immediately after **SNAModemInitialize**; otherwise it is preferable to call **SNAModemAddLink** as each port is initialized.

Syntax

```
            void SNAModemAddLink(
  MODEM_STATUS **ppModemStatus);
```

Parameters

ppModemStatus

The address of a pointer to a **MODEM_STATUS** structure that will be used for storing modem status information. The returned **MODEM_STATUS** structure will contain a link service name.

Remarks

Note

The IHV should declare a pointer to a **MODEM_STATUS** structure and pass its address to **SNAModemAddLink**.

Note

The **LSName** is initially the name of the link service, but may need to be altered for multiport link services.The IHV can replace the link service name returned in the **MODEM_STATUS** structure to differentiate between possible multiple connections through a single link service.

Note

The IHV should maintain the various input and output signal lines and the data flow frame counts in the returned **MODEM_STATUS** structure. The Microsoft Host Integration Server 2009 Modem Monitor application will periodically read and display the data stored in this **MODEM_STATUS** structure.

Note

Internally **SNAModemAddLink** increments the usage count of the shared memory, and signals the Host Integration Server 2009 Modem Monitor application that a new link has been added.

See Also

Reference

MODEM_STATUS

SNAModemInitialize

SNAModemDeleteLink

The **SNAModemDeleteLink** function should be called when a link is terminating to delete the resources associated with the link. The parameters passed in must correspond to those returned by a call to **SNAModemAddLink**.

Syntax

```
            void SNAModemDeleteLink(
  MODEM_STATUS *pModemStatus);
```

Parameters

pModemStatus

A pointer to a **MODEM_STATUS** structure that was passed to the **SNAModemAddLink** function used for storing modem status interface.

Remarks

All resources in the link service associated with the modem status are deleted, and they must not be accessed by the IHV code after calling this function.

See Also

Reference

MODEM_STATUS

SNAModemAddLink

SNAModemTerminate

The **SNAModemTerminate** function should be called once per link service process, at termination. The ideal place is **SNALinkTerminate**. If the link service supports a single link, it is appropriate to call **SNAModemDeleteLink** immediately before **SNAModemTerminate**. Otherwise it is better to call **SNAModemDeleteLink** as each link instance is terminated.

Syntax

```
void SNAModemTerminate();
```

See Also

Reference

SNALinkTerminate

SNAModemDeleteLink

SNA Perfmon API

This section provides reference material for the SNA performance monitoring structures and functions.

In This Section

- ADAPTERCOUNTER
- ADAPTERPERFDATA
- SNAInitLinkPerfmon
- SNAGetLinkPerfArea
- SNAGetPerfValues

ADAPTERCOUNTER

The **ADAPTERCOUNTER** structure represents an individual SNA Perfmon event that can be monitored, such as the total bytes transmitted. All of the data needed to display a single Perfmon event is stored in this structure.

Syntax

```
typedef struct adaptercounter
{
    ULONG count;
    ULONG type;
    LONG scale;
} ADAPTERCOUNTER;
```

Members

count

The count for a specific Perfmon event since startup of the link service. Each time a Perfmon event takes place, the count is incremented accordingly, based on the type of event being counted. This count is maintained by the link service.

type

The event type that is being monitored with this **ADAPTERCOUNTER**. The **type** member instructs Perfmon whether the **count** member represents a numeric counter such as number of connection failures, a rate such as throughput in bytes transferred per second, or a percentage. For suitable values, see the platform SDK documentation of **PERF_COUNTER_*** (for example, **PERF_COUNTER_COUNTER** or **PERF_COUNTER_RAWCOUNT**).

scale

The default scale to be used by the Perfmon application when displaying this event. The **count** member is scaled by 10 raised to the power of scale such that a scale member of –1 multiplies **count** by 0.1.

ADAPTERPERFDATA

The **ADAPTERPERFDATA** structure groups all of the ADAPTERCOUNTER structures for an SNA link service together into a single block. It also has a few fields used internally by the SNA Perfmon code. The SNA link driver should not change the first three members of this structure.

Syntax

```
typedef struct adapterperfdata
{
    ULONG inuse;
    ULONG ServiceNameIndex;
    ULONG FirstCounterIndex;
    ADAPTERCOUNTER TotalBytesReceived;
    ADAPTERCOUNTER TotalBytesTransmitted;
    ADAPTERCOUNTER TotalFramesReceived;
    ADAPTERCOUNTER TotalFramesTransmitted;
    ADAPTERCOUNTER SuccessfulConnects;
    ADAPTERCOUNTER ConnectionFailures;
    ADAPTERCOUNTER TotalBytesThroughput;
    ADAPTERCOUNTER TotalFramesThroughput;
    ADAPTERCOUNTER AdapterFailures;
    ADAPTERCOUNTER reserved[11];
    ULONG pad;
} ADAPTERPERFDATA;
```

Members

inuse

A flag that indicates that the link service is using this section of shared memory.

ServiceNameIndex

An index into an array of strings describing events that can be monitored by the Perfmon functions. These strings are stored in the registry under the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib key.

FirstCounterIndex

An index into an array of events that can be monitored by the Perfmon functions.

TotalBytesReceived

The number of data bytes received per second.

TotalBytesTransmitted

The number of data bytes transmitted per second.

TotalFramesReceived

The number of data frames received per second. A frame is an information structure recognized by one of the various protocols related to SNA. Frames contain multiple bytes of data.

TotalFramesTransmitted

The number of data frames transmitted per second.

SuccessfulConnects

The number of times since startup that a successful connection has been made.

ConnectionFailures

The number of times since startup that a connection has encountered an error condition.

TotalBytesThroughput

The total number of bytes flowing through Host Integration Server 2009 per second. This includes both incoming and outgoing bytes, and is a good indicator of how heavily your Host Integration Server is loaded.

TotalFramesThroughput

The total number of data frames flowing through Host Integration Server per second. This includes both incoming and outgoing frames, and is a good indicator of how heavily your Host Integration Server is loaded.

AdapterFailures

The number of times since startup that a network adapter has encountered an error condition.

reserved

An array of **ADAPTERCOUNTER** structures for future expansion.

pad

Padding.

SNAInitLinkPerfmon

The **SNAInitLinkPerfmon** function initializes the Perfmon data structures and code for an SNA link. The user defines the address of a handle and a void pointer that are passed as parameters to this function. This function returns values in these parameters that are then used by subsequent calls to the Perfmon code. The SNA link driver should not modify the parameters returned by this function.

Syntax

```
            void SNAInitLinkPerfmon(
    HANDLE *shrlockmutx,
        void **shrptr
);
```

Parameters

shrlockmutx

An address of a handle of a mutex used to protect a block of shared memory. This handle address is used when calling other Perfmon functions after initialization.

shrptr

The address of a pointer to a block of shared memory used by subsequent Perfmon functions.

SNAGetLinkPerfArea

The **SNAGetLinkPerfArea** function returns a pointer to the shared data area used by the Perfmon application to store the link statistics. The parameters are the returned values from **SNAInitLinkPerfmon**. The SNA link then maintains the **ADAPTERCOUNTER** members of the returned **ADAPTERPERFDATA** structure.

Syntax

```
            ADAPTERPERFDATA * SNAGetLinkPerfArea(
HANDLEshrlockmutx,
ADAPTERPERFDATA *shrptr
);
```

Parameters

shrlockmutx

The handle of a mutex used to protect a block of shared memory that will contain the **ADAPTERPERFDATA** structure for this SNA link. The address of this handle is returned by the **SNAInitLinkPerfmon** function.

shrptr

A pointer to a block of shared memory returned by **SNAInitLinkPerfmon** that will contain the **ADAPTERPERFDATA** structure used by the Perfmon functions for this SNA link.

See Also

Reference

ADAPTERPERFDATA

SNAInitLinkPerfmon

SNAGetPerfValues

The **SNAGetPerfValues** function is used to provide pointers to the *ServiceNameIndex* and *FirstCounterIndex* variables so that the Perfmon application knows where to get the descriptions of the performance counters from the registry. These variables are returned as members in the **ADAPTERPERFDATA** structure returned by the **SNAGetLinkPerfArea** function.

Syntax

```
            USHORT SNAGetPerfValues(
            int *pServiceNameIndex,int *pFirstCounterIndex
);
```

Parameters

pServiceNameIndex

A pointer to the **ServiceNameIndex** member of the **ADAPTERPERFDATA** structure.

pFirstCounterIndex

A pointer to the **FirstCounterIndex** member of the **ADAPTERPERFDATA** structure.

See Also

Reference

ADAPTERPERFDATA

SNAGetLinkPerfArea

Session Integrator Programmer's Reference

The following topics contain the COM reference material for the Session Integrator feature for Host Integration Server.

In This Section

Session Integrator COM Reference

Related Sections

LUA Programmer's Guide

Session Integrator Programmer's Guide

Microsoft.HostIntegration.SNA.Session

See Also

Other Resources

Network Integration Programmer's Reference

Session Integrator COM Reference

The following topics contain the COM reference material for the Session Integrator feature for Host Integration Server.

In This Section

IcomLU0 Interface

Icom3270 Interface

Reference

Microsoft.HostIntegration.SNA.Session

Related Sections

Session Integrator Programmer's Guide

LUA Programmer's Guide

See Also

Other Resources

Session Integrator Programmer's Reference

IcomLU0 Interface

The IcomLU0 interface allows access to an LU0 session.

Note
For more information about accessing IcomLU0 from managed text, see Microsoft.HostIntegration.SNA.Session.

Requirements

Type Library: COM3270 1.0 Type Library (siproxy.dll)

Platforms: Windows Server 2003 SP1 Standard and Enterprise Editions

See Also

Concepts

IcomLU0 Members

Other Resources

Session Integrator Programmer's Guide

IcomLU0 Members

The following table shows the IcomLU0 members.

Public Methods

Method	Description
IcomLUO.CreateSession Method	Creates a new LU0 session.
IcomLUO.SetProperty Method	Allows the comLU0 client to set property values for a session.
IcomLUO.GetProperty Method	Extracts both pre-defined and application-specific properties for the session.
IcomLUO.Connect Method	Connects a comLU0 client to an existing session.
IcomLUO.Disconnect Method	Disconnects the comLU0 client from a previously-connected session.
IcomLUO.Receive Method	Receives outbound data on a LU0 session.
IcomLUO.Send Method	Sends a complete inbound chain of data on an LU0 session.
IcomLUO.SendResponse Method	Sends a response or courtesy acknowledgement to the host.
IcomLUO.Online Method	Sets the LU0 session back in an on-line state after a call to Offline.
IcomLUO.Offline Method	Switches the LU0 session into an off-line state, which in turn causes the underlying SNA session to deactivate.

See Also

Concepts

IcomLU0 Interface

IcomLU0 Methods

The methods of the IcomLU0 interfaces are listed in the following table. For a complete list of the IcomLU0 members, see IcomLU0 Members.

Public Methods

Method	Description
IcomLUO.CreateSession Method	Creates a new LU0 session.
IcomLUO.SetProperty Method	Allows the comLU0 client to set property values for a session.
IcomLUO.GetProperty Method	Extracts both pre-defined and application-specific properties for the session.
IcomLUO.Connect Method	Connects a comLU0 client to an existing session.
IcomLUO.Disconnect Method	Disconnects the comLU0 client from a previously-connected session.
IcomLUO.Receive Method	Receives outbound data on a LU0 session.
IcomLUO.Send Method	Sends a complete inbound chain of data on an LU0 session.
IcomLUO.SendResponse Method	Sends a response or courtesy acknowledgement to the host.
IcomLUO.Online Method	Sets the LU0 session back in an on-line state after a call to Offline.
IcomLUO.Offline Method	Switches the LU0 session into an off-line state, which in turn causes the underlying SNA session to deactivate.

See Also

Concepts

IcomLU0 Interface

IcomLUO.CreateSession Method

Creates a new LU0 session.

Syntax

```
void CreateSession(
   string connectionSTR,
   short initType,
   ref System.Array data,
   int timeout,
   out object sessionHandle
)
```

Parameters

Parameter	Description
connectionSTR	NULL-terminated string that indicates the connection properties of the new session. The string is presented in a "PROPERTY=VALUE", space-delineated format. Connection property names and values are case insensitive. For more information about connection properties, see IcomLUO Session Properties.
initType	Contains the session initialization type. For more information, see the Comments section.
data	Pointer an array of type unsigned char that contains the INITSELF or SSCP logon message. Used only if *initType* contains INIT_INITSELF or INIT_LOGON.
timeout	The period of time in milliseconds to wait for the BIND and SDT commands to arrive. If the timeout expires before the SDT arrives the SNA server LU will be released and an error returned. Entering **0xffffffff** into *timeout* indicates an infinite wait time.
sessionHandle	When this method successfully returns, contains a pointer to the IUnknown interface to the comLU0 session object representing the underlying LU0 session. As along as a reference is kept to this interface, the session object will remain intact. This interface may be passed to the IcomLU0.Connect method to connect to the comLU0 object with the session. If no LU property is specified, comLU0 will select the best available LU assigned to the user account under which it is running.

Return Value

The following table describes the return codes for CreateSession.

Value	Description
S_OK	The LU0 session was successfully created. The LU session is active and ready to receive input.
CLU0_S_SSCP_ACTIVE	The LU0 session was successfully created. The SSCP session is active and ready to receive input. This return code is valid only when *initType* is set to **INIT_SSCP**.
CLU0_E_NEG_RESPONSE	The host or SNA server sent a negative response to the INITSELF. Optionally, the host or SNA server may have sent an unformatted logon command. This is true only if *initType* is set to **INIT_INITSELF** or **INIT_LOGON**.
CLU0_E_BADPARM	*connectionStr* contained an invalid property setting.

CLU0_E_NOFREELU	The LU specified in *luname* is an SNA server LU pool. The pool does not currently have any free LUs.
CLU0_E_LUINUSE	The LU specified in *luname* is an SNA server LU. This LU is currently being used by another application.
CLU0_E_LUNOTFOUND	The LU or pool name does not exist.
CLU0_E_TIMEDOUT	The session was not started within the specified timeout.
CLU0_E_SESSION_FAILED	The session failed to activate and is not connected to any TSS LU0 session. The application should attempt to create a new session using the same or different connection properties, or else connect to a different TSS session handle.
CLU0_E_ACCESSDENIED	The user account for the client does not have permission to use the requested LU or pool.
CLU0_E_ALREADY_CONNECTED	The comLU0 client is already connected to another session.
CLU0_E_SYSERROR	Failed due to an internal error.

Remarks

The following table contains the possible values for *initType*.

Name	**Value**	**Description**
INIT_BIND	0	Wait for an unsolicited BIND and SDT from the PLU.
INIT_SSCP	1	Wait for a BIND and SDT to arrive but allow access to the SSCP session for the application to send SSCP data and commands.
INIT_INITSELF	2	Wait for a BIND and SDT to arrive after sending the INITSELF command specified in *data*.
INIT_LOGON	3	Wait for a BIND and SDT to arrive after sending the UNFORMATTED SSCP logon message specified in *data*.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also
Concepts
IcomLU0 Methods
Other Resources
Using Session Integrator for LU0

IcomLUO.SetProperty Method

Allows the comLU0 client to set property values for a session.

Syntax

```
void SetProperty(
   string PropertyName,
    ref object value)
```

Parameters

Parameter	Description
PropertyName	The name of the property to be added or updated.
Value	The value of the specified property.

Return Value

Value	Description
S_OK	The requested property was added successfully.
CLU0_S_PROP_UPDATED	The existing property specified was updated successfully
CLU0_E_NOT_CONNECTED	The comLU0 client object is not connected to a session object through a call to the IcomLU0.Connect method.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU0_E_WAITING	Another thread has issued a Receive call for the specified comLU0 method which has not yet returned.
CLU0_E_BADPARAM	The specified property is a pre-defined session property, and therefore could not be updated.

Remarks

Property names are case-insensitive.

You cannot use SetProperty to modify any of the predefined connection properties. Instead, you can change only the properties defined with CreateSession.

See Also

Concepts

IcomLU0 Methods

Other Resources

Using Session Integrator for LU0

IcomLUO.GetProperty Method

Extracts both pre-defined and application-specific properties for the session.

Syntax

```
void SetProperty(
   string PropertyName,
   ref object value)
```

Parameters

Parameter	Description
PropertyName	The Name of the property to be extracted.
value	When the method returns, contains the value of the specified property.

Return Value

Value	Description
S_OK	The method completed successfully.
CLU0_E_NOT_CONNECTED	The comLU0 client object is not connected to a session object through a call to Icom3270.Connect.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU0_E_WAITING	Another thread has issued a Receive call for the specified comLU0 method, and has not yet returned.
CLU0_E_BADPARAM	The specified property has not been defined for this session.

Remarks

You can use getParameter, to determine the name of the pooled LU selected for the session when an LU0 session was originally created using an SNA Server pool name.

See Also

Concepts

IcomLU0 Methods

Other Resources

Using Session Integrator for LU0

IcomLUO.Connect Method

Connects a comLU0 client to an existing session.

Syntax

```
Void Connect(
   object sessionHandle
)
```

Parameters

Parameter	Description
sessionHandle	Pointer to an IUnknown that contains the session handle of the session to connect to.

Return Value

Value	Description
S_OK	The method has completed successfully.
CLU0_S_SSCP_ACTIVE	The comLU0 session is connected to an SSCP initiated session whose LU-LU session is not yet active. You should send the appropriate messages to the SSCP to solicit activation of the session.
CLU0_S_OFFLINE	The comLU0 session is connected to an SNA session that is currently offline. You should call IcomLU0.Online to activate the session.
CLU0_E_SESSION_FAILED	The underlying SNA session failed. You must disconnect and release the server session.
CLU0_E_ALREADY_CONNECTED	Another comLU0 client is connected to this session.
CLU0_E_SYSERROR	Connect failed due to an internal systems error.
E_NOINTERFACE	The session handle is not a valid IUnknown interface pointer.

Exceptions

Remarks

Once you connect successfully to a session, you are responsible for calling IcomLU0.Receive to provide processing time for outbound data.

You are guaranteed exclusive access to the session until you call IcomLU0.Disconnect.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

IcomLUO.Disconnect Method

Disconnects from a previously-connected session.

Syntax

```
void Disconnect()
```

Parameters

Return Value

Value	Description
S_OK	The method completed successfully.
CLU0_E_NOTCONNECTED	The comLU0 client is not connected to a session through a call to IcomLU0.Connect.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU_E_WAITING	Another thread has issued a receive call for this comLU0 method, and has not yet returned.
CLU0_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

After your call to Disconnect is complete, you must call Connect again in order to access the session. This is true regardless of whether your call to Disconnect was successful.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also
Concepts
IcomLU0 Methods
Other Resources
Using Session Integrator for LU0

IcomLUO.Receive Method

Receives outbound data on a LU0 session.

Syntax

```
void Receive(
   int timeout,
   ref int datasize,
   out int indication,
   out short seqno,
   ref System.Array data
)
```

Parameters

Value	Description
timeout	The period of time in milliseconds that the thread can wait for data to arrive. By setting *dataSize*, you can indicate if the application is willing to accept partial data after a timeout. Entering 0xffffffff into *howLong* indicates an infinite length of time.
datasize	The maximum amount of data that the application is willing to accept. If *dataSize* bytes of data are received before the timeout is compete, Receive will return the partial chain. When this method returns, contains the number of bytes present in the data buffer.
indication	One or more flags in a bitwise OR containing additional information about the outbound datastream. For more information, see the Remarks section.
seqno	When this method returns, contains the SNA sequence number of the chain. If NEG_RESPONSE is set in *indication*, *seqno* may instead contain the sequence number of the chain to which the host sent a response. The value returned in *seqno* may be used in IcomLU0.SendResponse to transmit a SNA response.
data	An array containing the data to receive.

Return Value

Value	Description
S_OK	A complete, or else the remainder of a partial, chain of data was received into the data buffer.
CLU0_S_PARTIAL_CHAIN	A partial chain of data was received into the data buffer.
CLU0_S_TIMEOUT	No data was received within the timeout specified. You should issue another Receive.
CLU0_E_SESSIONFAILURE	The LU0 session failed.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. The application should release the session handle.

CLU0_E_WAITING	Another thread has issued a Receive call for this method, and has not yet returned.
CLU0_E_SESSION_FAILED	The underlying SNA session failed, possibly due to a link outage or other transient failure. You must either disconnect and release the server session. Alternately, you may call IcomLU0.Offline to reset the session, and then call IcomLU0.Online to reactive the session.
CLU0_E_NOTCONNECTED	The comLU0 client is not connected to a session through a call to Icom3270.Connect.
CLU0_E_BADPARAM	One of the parameters contained an invalid value.
CLU0_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

Normally, Receive blocks until a complete chain of SNA data is available. However, the application can control the block through *howLong*, *maxData*, and *incompleteData*.

Receive returns only application-level data. Specifically, Receive will not return the SNA TH and RH headers.

The following table describes the possible values for *indication*.

Value	**Description**
SESSION_STARTED	One of the following: • The SSCP-initiated session has been activated. • A session that was reset by a CLEAR has been restarted by an SDT. • A session that previously received an UNBIUND has been reactivated by a BIND and SDT.
BEGIN_BRACKET	The host started a new bracket.
END_BRACKET	The host terminated the current bracket.
SEND	The host has given permission to send.
DATA_COMPLETE	The data represents a complete data chain or the end of a data chain.
DATA_INCOMPLETE	The data represents an incomplete data chain.
CANCEL	The last chain from the host was cancelled.
NO_RESPONSE	The application should not send a response to the data.
EXCEPTION_RESPONSE1/2	The application may send a negative response to reject the data, or a courtesy acknowledgement.
DEFINITE_RESPONSE1/2	The application must send a response to the data.
POS_RESPONSE	The host sent a positive response.

NEG_RESPONSE	The host sent a negative response.
EXR_REQUEST	The SNA server converted the host request into an exception request.
CHASE	The host requests that all outstanding responses be sent.
NORMAL_DATA	The data was received on the normal data flow.
EXPEDITED_DATA	The data was received on the expedited data flow.
APPL_DATA	The data is application (FMD) data.
FM_DATA	The data is Function Management (FMH) data.
LU_DATA	The data was received on the LU session.
SSCP_DATA	The data was received on the SSCP session.
CLEAR	The host has cleared the session.
QUIESCE	The host has quiesced the session.
SHUTDOWN	The host is shutting down the session.
RELEASE	The host cancelled the quiesce or shutdown state.
UNBIND	The host unbound the LU-LU session.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also
Concepts
IcomLU0 Methods
Other Resources
Using Session Integrator for LU0

IcomLUO.Send Method

Sends a complete inbound chain of data on an LU0 session.

Syntax

```
void Send(
   int hint,
   ref System.Array data,
    out short seqno)
```

Parameters

Parameter	Description
hint	A hint from the application regarding how the data is to be processed. For more information, see the remarks section.
data	The data to send.
seqno	When this method returns, contains the SNA sequence number of the chain. You can use the value returned by *seqno* to correlate any response the host may send later.

Return Value

Value	Description
S_OK	The data was sent successfully. If relevant, positive response was also received.
CLU0_S_MULTI_CHAIN	The session does not support multi-RU chains, but the data was larger than the RU size. comLU0 sent the data as a sequence of single RU chains.
CLU0_S_DEFINITE_RSP_MODE	comLU0 sent the data using DEFINITE_RESPONSE mode when EXCEPTION_RESPONSE or NO_RESPONSE was requested.
CLU0_S_EXCEPTION_RSP_MODE	comLU0 send the data using EXCEPTION_RESPONSE mode when DEFINITE_RESPONSE or NO_RESPONSE was requested.
CLU0_S_NO_RSP_MODE	comLU0 sent the data using NO_RESPONSE mode when DEFINITE_RESPONSE or EXCEPTION_RESPONSE was requested.
CLU0_E_NEG_RESPONSE	The host or SNA server sent a negative response to the DEFINITE_RESPONSE.
CLU0_E_NO_RSP_REQUESTED	No response was received from the host to a RQD request. You should call IcomLU0.Receive to determine the reason that he response was not received. For example, a CLEAR may have been received, or the session experienced an outage.
CLU0_E_BRACKED_NOT_ALLOWED	The session was between brackets but comLU0 was not allowed to start a new bracket. This occurred due to comLU0 receiving an SBI from the host.
CLU0_E_SESSION_FAILED	The underlying SNA session failed, possibly do to a link outage or other transient failure. You must disconnect and release the server session. Optionally, you may call IcomLU0.Offline to reset the session, and then call IcomLU0.Online to reactive the session.

CLU0_E_RECEIVE_IN_PROGRESS	The application has not completed receiving the last chain sent by the host. This is likely indicated by Receive returning the DATA_INCOMPLETE message. You should re-issue IcomLU0.Receive call to collect the remaining data and then call Send again.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU0_E_WAITING	Another thread has issued a Receive call for this method, which has not yet returned.
CLU0_E_SESSIONFAILURE	The LU0 session failed.
CLU0_E_NOTCONNECTED	The comLU0 client is not connected to a session through a call to Icom3270.Connect.
CLU0_E_SYSERROR	The send failed due to a system error.

Exceptions

Remarks

The SNA TH and RH are provided by comLU0 and must not be present in the data presented by the application.

The following table describes the possible values for *hint*.

Value	**Description**
END_BRACKET	comLU0 should end the current bracket.
PREPARE_TO_RECEIVE	The application is about to enter the receive state.
NO_RESPONSE	The application does not need a response from the host.
EXCEPTION_RESPONSE1/2	The application requires that the host send a negative response only.
DEFINITIE_RESPONSE1/2	The application requires that the host send a response to the data.
NORMAL_DATA	The application is sending the on the normal data flow.
EXPEDITED_DATA	The application is sending the data on the expedited data flow.
APPL_DATA	The data is application (FMD) data.
FM_DATA	The data is Function Management (FMH) data
LU_DATA	The application is sending the data on the LU session.
SCP_DATA	The application is sending the data on the SSCP session.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also

Concepts

IcomLU0 Methods

Other Resources

Using Session Integrator for LU0

IcomLUO.SendResponse Method

Sends a response or courtesy acknowledgement to the host.

Syntax

```
void SendResponse(
   int senseCode,
   int hint,
   short seqno
)
```

Parameters

Parameter	Description
senseCode	The sense code to send to the host, in Intel byte order. 0x00000000 indicates a positive response or a courtesy acknowledgement to exception response data.
hint	A hint to indicate the message flow on which the response is to be sent. Hint should be a bitwise combination of LU_DATA or SSCP_DATA and NORMAL_DATA or EXPEDITED_DATA.
seqno	The sequence number of the request to respond to. The value used in *seqno* is returned by IcomLU0.Receive.

Return Value

Value	Description
S_OK	The method sent the message successfully.
CLU0_E_SESSION_FAILED	The underlying SNA session failed, possibly due to a link outage or other transient failure. You must disconnect and release the server session. Optionally, you may issue a call to IcomLU0.Offline to reset the session, and then reactivate the session with a call to IcomLU0.Online.
CLU0_E_RECEIVE_IN_PROGRESS	The application has not completed receiving the last chain sent by the host. This may be indicated by Receive returning DATA_INCOMPLETE. The you should re-issue the IcomLU0.Receive call to collect the remaining data and then retry the call to SendResponse.
CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU0_E_WAITING	Another thread has issued a Receive call for this comLU0 method which has not yet returned.
CLU0_E_NOTCONNECTED	The comLU0 client is not connected to a session object through a call to Connect.
CLU0_E_SYSERROR	This method failed due to an internal error.

Exceptions

Remarks

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also
Concepts
IcomLU0 Methods
Other Resources
Using Session Integrator for LU0

IcomLUO.Online Method

Sets the LU0 session back in an on-line state after a call to Offline.

Syntax

```
void Online(
   short initType,
   ref System.Array data,
   int timeout)
```

Parameters

Parameter	Description
initType	Describes the session initiation type. For more information, see the Remarks section.
data	Contains the INITSELF or SSCP logon message, if necessary.
timeout	The period of time in milliseconds to wait for the BIND and SDT to arrive. If the timeout expires before the SDT arrives, the SNA server LU will be released and an error returned. 0xffffffff indicates an infinite timeout.

Return Value

Value	Description
S_OK	The LU0 session was successfully reactivated and the LU session is active and ready to receive input.
CLU0_S_SSCP_ACTIVE	The LU0 session was successfully reactivated and the SSCP session is active and ready to receive input. Valid only when *initType* is set to INIT_SSCP.
CLU0_E_NEG_RESPONSE	The host or SNA server sent a negative response to the INITSELF or unformatted logon command. Valid only if *initType* is set to INIT_INITSELF or INIT_LOGON
CLU0_E_BADPARAM	*connectionStr* contained an invalid property setting.
CLU0_E_NOFREELU	*luname* specified an SNA server LU pool, and no LUs are free in that pool.
CLU0_E_LUINUSE	*luname* specified an SNA server LU, and the LU is currently in use by another application.
CLU0_E_LUNOTFOUND	The LU or pool name does not exist.
CLU0_E_TIMEDOUT	The session was not started within the timeout specified.
CLU0_E_SESSION_FAILED	The underlying SNA session failed, possibly due to a link outage or other transient failure. You must disconnect and release the server session. Optionally, you may issue a call to Icom3270.Offline to reset the server, and then reactivate the session using a call to Icom3270.Online.

CLU0_E_SERVER_FAILURE	The TSS session is no longer valid. You should release the session handle.
CLU_E_WAITING	Another thread has issued a Receive call for this method, which has not yet returned.
CLU_E_SYSERROR	This method failed due to an internal error.

Exceptions

Remarks

Online will attempt to acquire the same SNA server LU, and therefore the same SNA server, used when the session was last on-line.

The following table describes the possible values for *initType*.

Name	Value	Description
INIT_BIND	0	Wait for unsolicited BIND and SDT from the PLU.
INIT_SSCP	1	Wait for a BIND and SDT to arrive but allow access to the SSCP session for the application to send SSCP data and commands.
INIT_INITSELF	2	Wait for a BIND and SDT to arrive after sending the INITSELF command specified in *data*.
INIT_LOGON	3	Wait for a BIND and SDT to arrive after sending the UNFORMATTED SSCP logon message specified in *data*.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

See Also
Concepts
IcomLU0 Methods
Other Resources
Using Session Integrator for LU0

IcomLUO.Offline Method

Switches the LU0 session into an off-line state, which in turn causes the underlying SNA session to deactivate.

Syntax

```
void Offline()
```

Parameters

Return Value

Value	Description
S_OK	The session has been successfully deactivated.
CLU0_E_WAITING	Another thread has issued a Receive call for this comLU0 method, which has not yet returned.
CLU_E_SERVER_FAILURE	The TSS session is no longer valid. The application should release the session handle.
CLU0_E_RECEIVE_IN_PROGRESS	The application has not yet completed receiving the last chain sent by the host. This may be indicated by Receive returning DATA_INCOMPLETE. You should re-issue the IcomLU0.Receive call to collect the remaining data, and then call IcomLU0.Offline again.
CLU0_E_SYSERROR	This method failed due to an internal error.

Exceptions

Remarks

After calling Offline, the client application can later reactivate the session using a call to Online.

Note that Offline releases the SNA server LU. Therefore, it is possible for another application to acquire the LU before your application calls Online again.

You can use Online to recover a session that has returned CLU0_E_SESSION_FAILED.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270 Interface

The Icom3270 interface allows access to an LU0 session.

Note
For more information about accessing IcomLU0 from managed text, see Microsoft.HostIntegration.SNA.Session.

Requirements

Type Library: COM3270 1.0 Type Library (siproxy.dll)

Platforms: Windows Server 2003 SP1 Standard and Enterprise Editions.

Icom3270 Members

The following table describes the Icom3270 members.

Public Methods

Method	Description
Icom3270.createSession Method	Creates a new 3270 session.
Icom3270.getProperty Method	Describes the properties of a session.
Icom3270.connect Method	Connects a com3270 client to an existing session.
Icom3270.disconnect Method	Disconnects from a session.
Icom3270.setCursorPosition Method	Sets the position of the cursor on the 3270 screen.
Icom3270.getCursorPosition Method	Retrieves the current cursor position as an offset of the 3270 display buffer.
Icom3270.sendKey Method	Sends one or more keystrokes to the Host session.
Icom3270.wait Method	Waits for the session to enter a state where input is allowed or the screen is modified.
Icom3270.getOIA Method	Returns a copy of the Operator Information Area (OIA) for the 3270 session.
Icom3270.getScreenSize Method	Returns the size, in rows and columns, of the current 3270 screen.
Icom3270.getField Method	Returns the starting position and length of the field containing the specified screen offset .
Icom3270.getNextField Method	Finds the starting position and length of the field located after the specified offset.
Icom3270.getPrevField Method	Finds the starting position and length of the field before the specified screen offset.
Icom3270.getFieldData Method	Extracts the data contents of the specified field.
Icom3270.setFieldData Method	Sets the data contents pf the specified field.
Icom3270.findFieldData Method	Searches the specified field for the specified data string.
Icom3270.getScreenData Method	Extracts the data contents of the 3270 screen.
Icom3270.setScreenData Method	Copies data characters, character attributes, and extended attributers to all or part of the screen.
Icom3270.findScreenData Method	Searches the screen for a specified data string.

See Also

Concepts

Icom3270 Interface

Icom3270 Methods

The methods of the Icom3720 interfaces are listed in the following table. For a complete list of the IcomLU0 members, see Icom3270 Members.

Public Methods

Method	Description
Icom3270.createSession Method	Creates a new 3270 session.
Icom3270.getProperty Method	Describes the properties of a session.
Icom3270.connect Method	Connects a com3270 client to an existing session.
Icom3270.disconnect Method	Disconnects from a session.
Icom3270.setCursorPosition Method	Sets the position of the cursor on the 3270 screen.
Icom3270.getCursorPosition Method	Retrieves the current cursor position as an offset of the 3270 display buffer.
Icom3270.sendKey Method	Sends one or more keystrokes to the Host session.
Icom3270.wait Method	Waits for the session to enter a state where input is allowed or the screen is modified.
Icom3270.getOIA Method	Returns a copy of the Operator Information Area (OIA) for the 3270 session.
Icom3270.getScreenSize Method	Returns the size, in rows and columns, of the current 3270 screen.
Icom3270.getField Method	Returns the starting position and length of the field containing the specified screen offset.
Icom3270.getNextField Method	Finds the starting position and length of the field located after the specified offset.
Icom3270.getPrevField Method	Finds the starting position and length of the field before the specified screen offset.
Icom3270.getFieldData Method	Extracts the data contents of the specified field.
Icom3270.setFieldData Method	Sets the data contents pf the specified field.
Icom3270.findFieldData Method	Searches the specified field for the specified data string.
Icom3270.getScreenData Method	Extracts the data contents of the 3270 screen.
Icom3270.setScreenData Method	Copies data characters, character attributes, and extended attributers to all or part of the screen.
Icom3270.findScreenData Method	Searches the screen for a specified data string.

Icom3270.createSession Method

The creaetSession method creates a new 3270 session.

Syntax

```
void CreateSession(
   string connectionStr,
   out object sessionHandle
)
```

Parameters

Parameter	Description
connectionStr	NULL-terminated string containing the connection properties of the new session. The string is presented in a space-delineated "PROPERTY=VALUE" format. Property names and values are case-insensitive. For more information, see Icom3270 Session Properties.
sessionHandle	When this method returns, contains a pointer to the IUnknown interface for the comSNA3270 session. This session represents the underlying SNA session. You will use *sessionHandle* in Icom3270.Connect to connect the com3270 object with the session.

Return Value

Value	Description
C3270_S_LINKINAC	The SNA session was successfully created, but the underlying SNA connection is not yet active. You should use Icom3270.wait to wait for the session to become active.
C3270_S_PUINAC	The SNA session was successfully created, but the SNA PU is not yet active. You should use Icom3270.wait to wait for the session to become active.
C3270_S_LUINAC	The SNA session was successfully created, but he LU is not yet active. You should use Icom3270.wait to wait for the session to become active.
C3270_S_TN3270E	The TN3270 session was successfully created in TN3270E, or RFC 1647, mode.
C3270_S_TN3270	The TN3270 session was successfully created in TN3270, or RFC 1576, mode.
S_OK	The SNA or local session was successfully created and the LU_SSCP session is active and ready to receive input.
C3280_E_BADPARAM	*connectionStr* contained an invalid property setting.
C3270_E_NOFREELU	*luname* specified an SNA Server LU pool. However, there are no LUs free in that pool.
C3270_E_LUINUSE	*luname* specified an SNA server LU pool. However, the LU is currently in use by another application.
C3270_E_LUNOTFOUND	The LU or pool name does not exist.
C3270_E_SYSERROR	The method failed due to an internal error.

C3270_E_ACCESSDENIED	The user account for the client does not have permission to use the requested LU or pool.
C3270_E_NOTFOUND	The remote system specified by the IP property in *connectionStr* could not be located.
C3270_E_REFUSED	The remote system specified by the IP and PORT properties in *connectionSTR* refused the connection string.

Exceptions

Remarks

The following list describes the location of the session, as determined by to the session type. A session type is also known as the MODE property.

- Local session - created on the client.
- SNA session - created on the SNA server machine that is configured with the requested LU or pool.
- TN3270 session - created on the COM server specified by the SERVER property.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getProperty Method

The getProperty method describes the properties of a session.

Syntax

```
void GetProperty(
   string PropertyName,
   out object value
)
```

Parameters

Parameter	Description
propertyName	The name of the property to be returned.
propertyValue	When this method returns, contains the value of the specified property.

Return Value

Value	Description
S_OK	The method completed successfully.
C3270_E_NOT_CONNECTED	The com3270 client is not connected to a session object through Icom3270.Connect.
C3270_E_INVALIDMODE	The property described in *propertyName* is not valid for the session MODE setting.

Exceptions

Remarks

You can use getProperty to determine the name of the pooled LU selected for the session when an SNA session was originally created using an SNA server pool name. For more information, see Supported com3270 Session Properties.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.connect Method

The connect method connects a com3270 client to an existing session.

Syntax

```
void Connect(
   object sessionHandle
)
```

Parameters

Parameter	Description
sessionHandle	Session handle of the session to connect to.

Return Value

Value	Description
S_OK	The method completed successfully.
C3270_E_ALREADY_CONNECTED	Another com3270 client is connected to the specified session.
C3270_E_SYSERROR	The method failed due to an internal error.
E_NOINTERFACE	The specified session handle is not a valid comSNA3270, COMTN3270, or comLocal3270 IUnknown interface pointer.

Exceptions

Remarks

Once you connect successfully to a session, you are responsible for calling Icom3270.wait in order to provide processing time for incoming 3270 data stream packets.

You are guaranteed exclusive access to the session until you call disconnect. Specifically, the screen buffers and screen size are guarnenteed not to change unless you call Icom3270.wait.

If connect completes successfully, you should then call Icom3270.wait. Calling wait allows you to process any unprocessed 3270 data and to ensure that the session is in an unlocked state. After calling wait, you can then update or read the display buffers or send keystrokes to the host application.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.disconnect Method

The disconnect method disconnects from a session.

Syntax

```
void Disconnect()
```

Parameters

Return Value

Value	Description
S_OK	The method completed successfully
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

After you call disconnect, even if the call fails, you must call Icom3270.connect again before accessing the session.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements

Subhead

Icom3270.setCursorPosition Method

The setCursorPosition sets the position of the cursor on the 3270 screen.

Syntax

```
void SetCursorPosition(
   ushort position
)
```

Parameters

Parameter	Description
pos	A 0-based offset within the screen buffer that describes the location of the cursor.

Return Value

Value	Description
S_OK	The method has completed successfully.
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_SESSIONBUSY	The 3270 session is busy. You may call Icom3270.wait to determine when input is allowed. Afterwards, you can attempt to call setCursorPosition again.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

After you call setCursor, you may call sendKey. Calling sendKey will write printable character, starting from the location specified by setCursor.

To determine the offset of a particular row and column character address, you can use getScreenSize to retrieve the number of columns in each screen row.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getCursorPosition Method

The getCursorPosition method retrieves the current cursor position as an offset of the 3270 display buffer.

Syntax

```
void GetCursorPosition(
   out ushort position
)
```

Parameters

Parameter	Description
position	When this method returns, contains a 0-based offset describing the current cursor position.

Return Value

Value	Description
S_OK	The method has completed successfully.
C3270_E_SESSIONBUSY	The 3270 session is busy. You may call wait to determine when input is allowed and getCurrentPosition can be retried.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

The cursor position may change as a result of a call to setCursorPosition, sendKey, or wait.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.sendKey Method

The sendKey method sends one or more keystrokes to the Host session.

Syntax

```
void SendKey(
   ushort count,
   ref System.Array keys
)
```

Parameters

Parameter	Description
count	The number of bytes in the buffer.
keys	A pointer to the buffer containing the keystrokes to send.

Return Value

Value	Description
S_OK	The method has returned successfully.
C3270_E_BADPARAM	One of the parameters is invalid.
C3270_E_SESSIONBUSY	The 3270 session is busy. You may call Icom3270.wait to determine when input is allowed before retrying sendKey.
C3270_E_SESSIONLOCKED	The 3270 session is locked due to a local lock condition. You may examine the OIA buffer to determine the reason for the error. Afterwards, you may want to send a RESET keystroke to unlock the keyboard before calling sendKey again or any other recovery action.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

sendKey emulates a user typing at the 3270 keyboard. You can use sendkey to send3270 functions, such as RESET, INSERT, or TAB, to the session. You can also send both single and double-byte graphic keys to the session.

Note that if you send an AID key, the host will ignore any subsequent keys in the buffer. As such, the AID key should be the last keystroke you send.

Graphic keys are represented by their EBCIDIC character balue, or two characters for DBCS in the Host code page. 3270 Function and AID keys are represented by the multi-byte EBCIDIK key codes described in the Microsoft SNA Server Windows HLLAPI Specification.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.wait Method

The wait method waits for the session to enter a state where input is allowed or the screen is modified.

Syntax

```
void Wait(
   uint howLong,
   int waitForUpdate
)
```

Parameters

Parameters	Value
howLong	A period of time, measured in units of 0.5 seconds, that the thread is willing to waif for input to be enabled or the screen to be updated. 0xffffffff indicates that the thread should wait indefinitely.
waitForUpdate	If false, this method will return as soon as the session is in an input allowed state. The session returns immediately if the session is currently in an input allowed state. For more information, see the Remarks section.

Return Value

Value	Description
S_OK	The session is available for input.
C3270_S_SIZECHANGED	The session is available for input, but the screen size was modified during the invocation of wait. You should gall getScreenSize to determine the new screen size.
C3270_E_SESSIONBUSY	The 3270 session is still busy, but the time-out period specified by howLong expired. You should perform any necessary processing before calling wait again.
C3270_E_SESSSIONLOCKED	The 3270 session is locked due to a local lock condition. You should examine the OIA buffer to determine the reason for the error. You may also send a RESET keystroke to unlock the keyboard before calling wait again or performing any other recovery action.
C3270_E_SESSIONFAILURE	The 3270 session failed. Either the PLU_SLU or SSCP session was deactivated while the wait was in progress. You should examine the session status in the OIA for the session and take appropriate recovery action.
C3270_E_SYSERROR	The method failed due to an internal error.

Remarks

Calling wait allows the session to process messages from the host when the application is active, connected to the host, and waiting for data.

You should set waitForUpdate to true when the Host unlocks the keyboard and sends screen updates in separate operations. In particular, you should do this on the SSCP session, where input is enabled on receipt of an SNA response to data from the client. The reply data is dent on a subsequent message.

Example

Icom3270.getOIA Method

The getOIA method returns a copy of the Operator Information Area (OIA) for the 3270 session.

Syntax

```
void GetOIA(
   out System.Array oiaBuffer
)
```

Parameters

Parameter	Description
oiaBuffer	When this method returns, contains the buffer into which the current OIA is copied. For more information, see the Remarks section.

Return Value

Value	Description
S_OK	The method has completed successfully.
C3270_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

The OIA is a bit map that provides information regarding the status of the current screen. You may use the OIA to determine the ownership of the session, and the reason for input-inhibited errors.

As defined by COM, it is your responsibility to release the memory of the returned SAFEARRAY structures.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getScreenSize Method

The getScreenSize method returns the size, in rows and columns, of the current 3270 screen.

Syntax

```
void GetScreenSize(
   out ushort rows,
   out ushort cols
)
```

Parameters

Parameter	Description
rows	When this method returns, contains the number of rows on the current screen.
cols	When this method returns, contains the number of columns on the current screen.

Return Value

Value	Description
S_OK	The method has returned successfully.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

You should call getScreenSize immediately after Icom3270.wait to determine whether the screen size has changed as a result of a Host application command.

You should not attempt to access or modify the screen buffers returned by the Icom3270.connect beyond the limits of the current screen size.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getField Method

The getField method returns the starting position and length of the field containing the specified screen offset.

Syntax

```
void GetField(
   ref ushort position,
   out ushort length
 )
```

Parameters

Parameter	Description
position	The 0-based screen offset of a character in the desired field. Note that the calculation used by getField considers the field attribute character part of the field. The field attribute character immediately precedes the field data. When this method returns, contains the 0-based offset of the first data position of the field.
length	When this method returns, contains the length of the desired field, excluding the field attribute character. Possible values range between 0 and (screen size - 1).

Return Value

Value	Description
S_OK	The method has completed successfully.
C3270_E_INVALIDPOS	The screen position specified is greater than the maximum character position for the current screen size.
C3270_E_UNFORMATTED	The screen is unformatted. Therefore, the specified field does not exist.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session object through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getNextField Method

The getNextField method finds the starting position and length of the field located after the specified offset.

Syntax

```
void GetNextField(
   ref ushort position,
   ushort fieldType,
   out ushort length
)
```

Parameters

Parameter	Description
position	The 0-based screen offset from which to start the search. When this method returns, contains the 0-based offset of the first data position of the following field.
fieldType	The type of field requested. For more information, see the Remarks section.
length	When this method returns, contains the length of the desired field, excluding the field attribute character. Possible values are 0 through (screen size - 1).

Return Value

Value	Description
S_OK	The method has returned successfully.
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_UNFORMATTED	The screen is unformatted. Therefore, the specified field does not exist.
C3270_E_NOTFOUND	The specified field could not be found.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icon3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

For the purpose of this search, the field attribute character is considered part of the field. The field attribute character immediately precedes the field data.

The following table describes the possible values for fieldType.

Value	Description
0	Either protected or unprotected.
1	Unprotected field only.
2	Protected field only.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getPrevField Method

The getPrevField finds the starting position and length of the field before the specified screen offset.

Syntax

```
void GetPrevField(
   ref ushort position,
   ushort fieldType,
   out ushort length
)
```

Parameters

Parameter	Description
position	The 0-based screen offset to start searching from. When this method returns, contains the 0-based offset of the first data position in the previous field.
fieldType	The type of field requested. For more information, see the Remarks section.
length	When this method returns, contains the length of the desired field, excluding the field attribute character. Possible values are 0 through (screen size - 1).

Return Value

Value	Description
S_OK	The method has returned successfully
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_UNFORMATTED	The screen is unformatted. Therefore, the specified field does not exist.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

For the purpose of getPrevField, the field attribute character is considered part of the field. The field attribute character immediately precedes the field.

The following table describes the possible values for *fieldType*.

Value	Description
0	Either protected or unprotected
1	Unprotected field only
2	Protected field only

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getFieldData Method

Extracts the data contents of the specified field.

Syntax

```
void GetFieldData(
   ushort position,
   ushort dataRequested,
   ushort maxLen,
   out System.Array dbuf,
   out System.Array abuf,
   out System.Array eabuf
)
```

Parameters

Parameter	Description
position	The 0-based screen offset of a character in the specified field.
dataRequested	A bitwise combination describing the data requested. For more information, see the Remarks section.
maxLen	The maximum number of .screen positions requested. 0 indicates a request for the entire field.
dbuf	When this method returns, contains the screen data buffer data, if necessary.
abuf	When this method returns, contains the screen character attribute buffer data, if necessary.
eabuff	When this method returns, contains the screen extended attribute buffer data, if necessary.

Property Value/Return Value

Exceptions

Remarks

getFieldData does not extract the Field Attribute character.

You may request any combination of the displayable characters, character attributes, and extended attributes of the field.

You are responsible for releasing the SAFEARRAYs that the method returns the specified data in.

For the purpose of getFieldData the field attribute character, which immediately precedes the field data, is considered part of the field.

The following table describes the possible values for *dataRequested*.

Value	Description
1	Display buffer data
2	Character attribute buffer data
4	Extended attribute buffer data

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.setFieldData Method

The setFieldData method sets the data contents of the specified field.

Syntax

```
void SetFieldData(
   ushort position,
   ushort length,
   int overwriteProtected,
   ref System.Array dbuf,
   ref System.Array abuf,
   ref System.Array eabuf
)
```

Parameters

Parameter	Value
position	The 0-based screen offset of a character in the desired field.
length	The length of the data to copy.
overwriteProtected	If true, allows you to write data to a protected file. Otherwise, attempting to write to a protected field will cause an error.
dbuf	The data to copy to the screen data buffer data. NULL indicates that you do not want to alter the screen data.
abuf	The data to copy to the screen character attribute buffer data. NULL indicates that you do not want to alter the character attribute buffer.
eabuf	When this method returns, contains the data to copy to the screen extended attribute buffer data. NULL indicates that you do not want to alter the extended attribute buffer.

Return Value

Value	Description
S_OK	The method completed successfully.
C3270_S_TRUNCATED	The copy extended past the end of the field. Therefore, the extra data was ignored.
C3270_E_INVALIDPOS	The screen position specified is greated than the maximum character position for the current screen size.
C3270_E_INVALIDDATA	The bounds of any of the non-NULL SAFEARRA parameters were not identical.
C3270_E_UNFORMATTED	The screen is unformatted. As such, the specified field does not exist.
C3270_E_SESSIONBUSY	The 3270 session is busy. Call Icom3270.wait to determine when input is allowed so that you may call this method again.
C3270_E_SESSIONLOCKED	The 3270 session is loced due to a local lock condition. Examine the OIA buffer to determine the reason for the error. Also, you may also send a RESET keystroke to unlock the keyboard before calling this method or taking any other recovery action.

C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

You cannot use setFieldData to modify the Field attribute character.

For the purpose of setFieldData, the field attribute character is considered part of the field. The field attribute character immediately precedes the field data.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.findFieldData Method

The findFieldData method searches the specified field for the specified data string.

Syntax

```
void FindFieldData(
   ref ushort position,
   ushort length,
   ref System.Array dbuf
)
```

Parameters

Parameter	Description
pos	The 0-based screen offset of a character in the field to search. When this method returns, contains the screen offset of the start of the data string, if the string was found.
length	The length of the data to search on
dbuf	Array containing the data to search on.

Return Value

Value	Description
S_OK	The method has completed successfully.
C3270_S_TRUNCATED	The copy extended past the end of the field. The extra data was ignored.
C3270_E_UNFORMATTED	The screen is formatted; therefore, the specified field does not exist.
C3270_E_NOTFOUND	The specified data string could not be found.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

For the purpose of findFieldData, the field attribute character is considered part of the field. The field attribute character immediately precedes the field data.

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.getScreenData Method

The getScreenData method extracts the data contents of the 3270 screen.

Syntax

```
void GetScreenData(
   ushort position,
   ushort dataRequested,
   ushort maxLen,
   out System.Array dbuf,
   out System.Array abuf,
   out System.Array eabuf
)
```

Parameters

Parameter	Description
position	The 0-based screen offset of the first character requested.
dataRequested	A bitwise combination describing the requested data. For more information, see the Remarks section.
maxLen	The maximum number of .screen position requested. Setting maxLen to 0 requests the remainder of the screen.
dbuf	When this method returns, contains the screen data buffer data, if requested.
abuf	When this method returns, contains the screen character attribute buffer data, if requested.
eabuf	When this method returns, contains the screen extended attribute buffer data, if requested.

Return Value

Value	Description
S_OK	The method has returned successfully
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

You may request any combination of the displayable characters, character attributes, and extended attributes, of the screen.

Note that the returned data is contained in one or more SAFEARRAYS. You are responsible for releasing the SAFEARRAYS after processing.

The following table describes the possible values of dataRequested.

Value	Description
1	Display buffer data
2	Character attribute buffer data

4	Extended attribute buffer data

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.setScreenData Method

The **setScreenData** method copies data characters, character attributes, and extended attributers to all or part of the screen.

Syntax

```
void SetScreenData(
   ushort position,
   ushort length,
   int overwriteProtected,
   ref System.Array dbuf,
   ref System.Array abuf,
   ref System.Array eabuf
)
```

Parameters

Parameter	Description
position	The 0-based screen offset of the screen position to start copying.
length	The length of the data to search on.
overwriteProtected	true to allow the application to write data to a protected field; otherwise attempting to write to a protected field will cause an error.
dbuf	The data to copy to the screen data buffer data. NULL indicates that you do not want to alter the data buffer.
abuf	The data to copy to the screen character attribute buffer data. NULL indicates that you do not want to alter the character attribute buffer.
eabuf	When this method returns, contains the data to copy to the screen extended attribute buffer data. NULL indicates that you do not want to alter the extended attribute buffer.

Return Value

Value	Description
S_OK	The method has returned successfully
C3270_E_TRUNCATED	The copy extended past the end of the buffer screen. The extra data was ignored.
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_INVALIDDATA	The bounds of any non-NULL SAFEARRAY parameters were not identical.
C3270_E_SESSIONBUSY	The 3270 session was busy. You should call Icom3270.wait to determine when input is allowed and this method can be retried.
C3270_E_SESSIONLOCKED	The 3270 session is locked to to a local lock condition. You should examine the OIA buffer to determine the reason for the lock. You may also send a RESET keystroke to unlock the keyboard before calling this method again.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.

C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks
Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Icom3270.findScreenData Method

The findScreenData method searches the screen for a specified data string.

Syntax

```
void FindScreenData(
   ref ushort position,
   ushort length,
   ref System.Array dbuf
)
```

Parameters

Parameter	Description
position	on input, the 0-based screen offset describing the location on which to begin the search. On a successful return, contains the screen offset of the start of the data string.
length	The length of the array containing the data to search on.
dbuf	An array containing the data to search on.

Return Value

Value	Description
S_OK	The method has returned successfully
C3270_E_INVALIDPOS	The specified screen position is greater than the maximum character position for the current screen size.
C3270_E_NOTFOUND	The specified data string could not be found.
C3270_E_NOTCONNECTED	The com3270 client is not connected to a session through a call to Icom3270.connect.
C3270_E_SYSERROR	The method failed due to an internal error.

Exceptions

Remarks

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Reference

The following section describes the data types and context properties for the Client-Based BizTalk adapter for WebSphere MQ.

In This Section

Data Type

Context Properties

See Also

Other Resources

Messaging Programmer's Reference

Data Type

Header properties in MQSeries messages are data structures contained in the message itself. The adapter automatically validates and converts certain values in MQSeries message headers when sending and receiving messages.

The following table describes the MQSeries data types and their validation and conversion.

MQSeries Data Type	Validation and Conversion
MQLONG	MQSeries performs the validation. Converts to a long integer. Values that are not valid prevent the message from going to the MQSeries queue.
MQCHAR	Converts to a string.
MQBYTE	Converts to a string that contains the characters 0-9, and a-f or A-F, representing the hexadecimal value of the number.

Many of the MQSeries properties are 32-bit (4-byte) unsigned integers. Because uint is not a Common Language Specification (CLS)-compliant type, you must assign them to object types before using them in .NET methods. For more information about CLS-compliant types, see "What Is the Common Language Specification?" in .NET Framework Help.

See Also
Other Resources
Client-Based BizTalk Adapter for WebSphere MQ Programmer's Reference

Context Properties

These context properties are only meaningful from BizTalk Server concepts and are available to be used by BizTalk applications.

In This Section

BizTalk-Specific Properties

Advanced End-Point Configuration Properties

MQSeries Header Properties

See Also

Other Resources

Network Integration Programmer's Reference

BizTalk-Specific Properties

These context properties are only meaningful from BizTalk Server concepts and are available to be used by BizTalk applications.

Name	Type	Receive and/or Send	Description
URI	String	Receive and send	The URI defines the end-point for receive locations and send ports. The URI for receive locations and send ports for MQSC Adapter are in the following format: mqsc://<CHANNELNAME>/<TRANSPORTTYPE>/ <CONNECTIONNAME>/<QUEUEMANAGERNAME>/<QUEUENAME> Receive example: mqsc://MYCHANNEL/tcp/MQSERVER(1414)/QM1/RECVQ Send example: mqsc://MYCHANNEL/tcp/MQSERVER(1414)/QM2/SENDQ
BizTalk_CorrelationID	String	Receive	Use this property to have the MQSeries server generate a correlation identifier for use with the message.

Advanced End-Point Configuration Properties

The following topics describe the Advanced End-Point configuration properties for the Client-Based BizTalk Adapter for WebSphere MQ.

In This Section

MQSeries.MQSPropertySchema Properties

MQSeriesEx.MQSPropertySchema Properties

See Also
Other Resources
Context Properties

MQSeries.MQSPropertySchema Properties

The MQSC Adapter exposes the following context properties that are not related to MQSeries Message Descriptor or other MQSeries header structures that can be used in your BizTalk applications. These are part of the MQSeries.dll property schema assembly (MQSeries.MQSPropertySchema) that is deployed in the BizTalk Management database from the server-based MQSeries Adapter. The same property schema is used by the MQSC (client-based) adapter.

Name	Type	Receive and/or Send	Description
CompleteMessage	String	Receive	Set MQSeries to assemble segmented messages or to get the message as is. Use NoAction to read messages from the MQSeries queue without enabling segmentation. Use CompleteMessage to have MQSeries assemble segmented messages before passing them on to the adapter. Default: No Action This is not applicable to the MQSC Adapter. Used only for the Server-Based MQSeries Adapter in 'dynamic receive' type scenarios.
DataConversion	String	Receive	Character set to which the message should be converted to when retrieving messages from the MQSeries Queue. If this property is set to a value other than 'None', the adapter sets the MQGMO CONVERT option when performing an MQGet. None - Do not convert. UCS-2 and UTF-16 - Convert to these character sets. MQSeries does not distinguish between them. UTF-8 - Convert to the UTF-8 character set. Default: None Note – Not supported in this pre-release.
Ordered	String	Receive	Specify Yes to maintain the order of the messages as they are received from the MQSeries queue and submitted to the BizTalk Server Message Box. For the send side, the adapter sends the message to the queue in the same order that it receives it from the message box. Specify No to not maintain message order. Note – For send side ordering, if you are not using Orchestration, you must enable 'Ordered Delivery' in the 'Transport Advanced Options' in the send port configuration. Note – For receive, if you are using Orchestration, you must also set the Ordered Delivery property to True in your orchestration for this receive location. Note – If ordering is enabled, the adapter switches to single-thread mode and uses synchronous-mode delivery of messages into BizTalk Server. This results in performance degradation, so unless you require ordered delivery, it is not recommended to enable this feature. Default: No
SegmentationAllowed	String	Send	Set this to Yes to tell MQSeries Queue Manager to create segmented messages when submitting large messages to MQSeries Queues. Default: No
SSOAffiliateApplication	String	Send	Sets the Single Sign-On (SSO) Affiliate Application name. You use the user ID and password from SSO for the MQMD_UserIdentifier, and the MQIIH_Authenticator (or MQCIH_Authenticator) property respectively. Default: Blank

WaitInterval	Int	Receive	When performing MQGet, specify the Wait Interval MQGMO option in seconds by setting this property. If there are no messages in the queue, the client will continue waiting for messages in the Queue without closing the connection. Unit – Seconds Default – 3
TransactionSupported	String	Receive and Send	When set to Yes, the adapter begins a Microsoft Distributed Transaction Coordinator (DTC) transaction between BizTalk Server and MQSeries. This guarantees once and only once delivery of messages and prevents loss of messages. Setting this option to Yes means that WebSphere MQ Extended Transactional Client (Extended-Client) is used on the BizTalk Server computer by the adapter. When set to No, there could be message duplication. In this case, the adapter uses the non-transactional WebSphere MQ Client (Base-Client) for integration with MQSeries. Default: Yes

MQSeriesEx.MQSPropertySchema Properties

This contains additional context properties that are applicable only to the MQSC adapter (client-based MQSeries Adapter) for receive location and send port configurations. They are not applicable to the server-based adapter. These properties are associated with the channel configuration.

Name	Type	Receive and/or Send	Description
Channel_HeartBeat	unsignedInt	Receive and Send	The time between checks to verify if the client-server connection is working. Unit – Seconds Default - 300
Channel_UserId	String	Receive and Send	Set MQSeries to assemble segmented messages or to get the message as is. Use NoAction to read messages from the MQSeries queue without enabling segmentation. Use CompleteMessage to have MQSeries assemble segmented messages before passing them on to the adapter. Default: NoAction
Channel_Password	String	Receive and Send	The password may be used by the MCA when attempting to initiate a secure LU 6.2 session with a remote MCA. The initial value is null. This is an optional property.
Channel_SslCipherSpecification	String	Receive and Send	This defines a single CipherSpec for an SSL connection that will be used by the end-point configured in the adapter. Both ends of a WebSphere MQ SSL channel definition must include the attribute and the value specified here should match the name specified on the server end of the channel. The value is a string with a maximum length of 32 characters. This is required only when SSL is configured for the MQSeries Client to remote Queue Managers communication.
Channel_SslClientAuthentication	String	Receive and Send	This property determines whether the channel needs to receive and authenticate an SSL certificate from an SSL client. This is Optional by default. If two-way authentication (client/server) is required, this property should be set to Required. Before doing this, you must have configured SSL in MQSeries to enable client/server authentication so that the SSL client can send a valid certificate for authentication to succeed. Default: Optional
Channel_SslPeerName	String	Receive and Send	The property is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of a WebSphere MQ channel. If the DN received from the peer does not match this value, the channel does not start. This is required only when SSL is configured for the MQSeries Client to Queue Managers communication.

MQSeries Header Properties

MQSC Adapter provides a set of context properties, specific to MQSeries, for use in your applications. You can use these properties in pipeline components, in your orchestrations and in filter expressions. For easy programmatic access, MQMD, MQXQH, MQCIH and MQIIH header structures can be directly accessed using these context properties.

All other MQSeries header structures (example: MQRFH) are supported by the MQSC Adapter. However, to access these headers, you need to do so with custom pipeline components and retrieve them from the body of the message. If you are setting them in the outbound message, then the pipeline component is responsible for ensuring that the message is constructed correctly.

To assign MQSeries context properties to a message destined to a send port that is bound to MQSC Adapter, use the message assignment operator and specify one of the available context properties in the MQSeries namespace.

The following is an example of setting the MQSeries MQMD_UserIdentifier property:

```
Message_2(MQSeries.MQMD_UserIdentifier) = "MeMyselfAndI";
```

You must obtain enumerated values from the C programming language header files included with the IBM MQSeries SDK. You can find these files in the Program Files\IBM\WebSphere MQ\Tools\c\include folder. These files define the values to use when setting or reading MQSeries context property values.

Hexadecimal string values are character strings representing binary values. They do not have a prefix such as 0x. They contain digits from 0 through 9 and letters from "a" through "f" or "A" through "F". The adapter ignores white space in them.

For more information about these properties, see the IBM WebSphere MQ documentation.

In This Section

Message Descriptor Properties

Additional MQSeries Related Properties

See Also

Other Resources

Context Properties

Message Descriptor Properties

The following table shows the complete set of available Message Descriptor (MQMD structure) properties and their corresponding types and values. These are part of the MQSeries.dll assembly that is deployed with the server-based MQSeries Adapter. The same assembly is used by the MQSC Adapter.

Name	Type	Length	Value
MQMD_AccountingToken	String	64	Hexadecimal string
MQMD_ApplIdentityData	String	32	Hexadecimal string
MQMD_ApplOriginData	String	4	String Default: space
MQMD_BackoutCount	unsigned int	4	Number Read only Default: 0
MQMD_CodedCharSetId	unsigned int	4	Number Default: 0
MQMD_CorrelId	String	48	Hexadecimal string
MQMD_Encoding	unsigned int	4	Number Use header file value. Default: 0
MQMD_Expiry	unsigned int	4	Number
MQMD_Feedback	unsigned int	4	Number Use header file value. Default: 0
MQMD_Format	String	8	String If set to MQXMIT, makes sure that the MQXQH properties have values.
MQMD_GroupID	String	48	Hexadecimal string
MQMD_MsgFlags	unsigned int	4	Number Use header file value. Default: 0
MQMD_MsgId	String	48	Hexadecimal string
MQMD_MsgSeqNumber	unsigned int	4	
MQMD_MsgType	unsigned int	4	Number Use header file value.
MQMD_Offset	unsigned int	4	
MQMD_OriginalLength	unsigned int	4	
MQMD_Persistence	unsigned int	4	Number Use header file value.
MQMD_Priority	unsigned int	4	Number

MQMD_PutApplName	string	28	String Default: space
MQMD_PutApplType	unsigned int	4	Number Use header file value. Default: 0
MQMD_PutDate	string	8	Date
MQMD_PutTime	string	8	Time
MQMD_ReplyToQ	string	48	String Default: space
MQMD_ReplyToQMgr	string	48	String Default: space
MQMD_Report	unsigned int	4	Number Use header file value.
MQMD_UserIdentifier	string	12	String Contains the user identifier when you use the SSOAffiliateApplication property.

When receiving messages directly from MQSeries transmission queues, BizTalk Adapter for MQSeries formats the transmission queue header properties (the MQXQH data structure) and places them in their corresponding context properties. When sending messages directly to MQSeries transmission queues, the header properties are formatted and assigned values from the corresponding context properties only if the MQMD_Format property has a value of MQXMIT. The following table describes the properties.

Name	Type	Length	Value
MQXQH_RemoteQMgrName	String	48	string
MQXQH_RemoteQName	String	48	string

Together with the properties listed earlier in this topic, the adapter populates the following Message Descriptor values following the same rules. The adapter prefixes these property names with MQXQH_ instead of MQMD_, but otherwise they map directly to those properties defined in the Message Descriptor table:

- MQXQH_MsgDesc_AccountingToken
- MQXQH_MsgDesc_ApplIdentityData
- MQXQH_MsgDesc_ApplOriginData
- MQXQH_MsgDesc_BackoutCount
- MQXQH_MsgDesc_CodedCharSetId
- MQXQH_MsgDesc_CorrelId
- MQXQH_MsgDesc_Encoding
- MQXQH_MsgDesc_Expiry
- MQXQH_MsgDesc_Feedback

- MQXQH_MsgDesc_Format
- MQXQH_MsgDesc_MsgId
- MQXQH_MsgDesc_MsgType
- MQXQH_MsgDesc_Persistence
- MQXQH_MsgDesc_Priority
- MQXQH_MsgDesc_PutApplName
- MQXQH_MsgDesc_PutApplType
- MQXQH_MsgDesc_PutDate
- MQXQH_MsgDesc_PutTime
- MQXQH_MsgDesc_ReplyToQ
- MQXQH_MsgDesc_ReplyToQMgr
- MQXQH_MsgDesc_Report
- MQXQH_MsgDesc_UserIdentifier

Additional MQSeries Related Properties

There are additional MQSeries-related properties included in the property schema and available for use within your BizTalk Server application. These are applicable when dealing with CICS or IMS applications. The following table lists these properties.

Name	Type	Length	Value
MQCIH_AbendCode	String	4	
MQCIH_ADSDescriptor	unsigned int	4	
MQCIH_AttentionId	string	4	
MQCIH_Authenticator	string	8	Set to the SSO password when you use the SSOAffiliateApplication property.
MQCIH_CancelCode	string	4	
MQCIH_CompCode	unsigned int	4	
MQCIH_ConversationalTask	unsigned int	4	
MQCIH_CursorPosition	unsigned int	4	
MQCIH_ErrorOffset	unsigned int	4	
MQCIH_Facility	string	16	Hexadecimal string
MQCIH_FacilityKeepTime	unsigned int	4	
MQCIH_FacilityLike	String	4	
MQCIH_Flags	unsigned int	4	
MQCIH_Format	String		
MQCIH_Function	String	4	
MQCIH_GetWaitInterval	unsigned int	4	
MQCIH_LinkType	unsigned int	4	
MQCIH_NextTransactionId	String	4	
MQCIH_OutputDataLength	unsigned int	4	
MQCIH_Reason	unsigned int	4	
MQCIH_ReplyToFormat	String		
MQCIH_ReturnCode	unsigned int	4	
MQCIH_StartCode	String	4	
MQCIH_TaskEndStatus	unsigned int	4	

MQCIH_TransactionId	String	4	
MQCIH_UOWControl	unsigned int	4	
MQIIH_Authenticator	String	8	Set to the SSO password when you use the SSOAffiliateApplication property.
MQIIH_CommitMode	String		
MQIIH_Flags	unsigned int	4	
MQIIH_Format	String		
MQIIH_LTermOverride	String	8	
MQIIH_MFSMapName	String	8	
MQIIH_ReplyToFormat	String		
MQIIH_SecurityScope	String		
MQIIH_TranInstanceId	String	32	Hexadecimal string
MQIIH_TranState	String		

Administration and Management Programmer's Reference

Host Integration Server 2009 included the following Windows Management Interface (WMI) providers to enable scripted management.

For general information about programming for WMI providers, see Administration and Management Programmer's Guide section of the SDK.

For sample code illustrating WMI providers, see Administration and Management Samples.

In This Section

Configuration Provider WMI Programmer's Reference

The Configuration Provider WMI Programmer's Reference describes the classes you can use to monitor the configuration of Host Integration Server 2009.

In This Section

- wmiHIS WMI Provider Classes

wmiHIS WMI Provider Classes

Microsoft® Host Integration Server provider supplies information regarding the configuration of Host Integration Server. As an instance provider, the wmiHIS provider implements the standard **IWbemProviderInit** interface and the following **IWbemServices** methods:

- **CreateInstanceEnumAsync**
- **DeleteInstanceAsync**
- **GetObjectAsync**
- **PutInstanceAsync**

For more information on **IWbemProviderInit** and **IWbemServices**, see "COM API for WMI" in the MSDN Library at http://msdn.microsoft.com/library.

You can access the WmiHIS provider classes in the \root\MicrosoftHIS namespace.

Class	Description
MsHis_Locale	Queries for locale support.
MsHis_CodePage	Queries for code page support.

MsHis_Locale

The **MsHIS_Locale** class is used to query for locale support.

The following syntax is simplified from MOF code.

Syntax

```
class MsHis_Locale : MsHis_Config
{
   uint32 ID;
   string Name;
   boolean Available;
};
```

Properties

ID

Data Type: **uint32**

Qualifiers: **Key**

Access Type: Read-Only

The locale identifier. Used for internal reference.

Name

Data Type: **String**

Access Type: Read-Only

The locale name.

Available

Data Type: **Boolean**

Access Type: Read-Only

true if the locale is available on the system; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

wmiHIS WMI Provider Classes

Administration and Management Programmer's Guide

MsHis_CodePage Class

The **MsHIS_CodePage** class is used to query for code page support.

The following syntax is simplified from MOF code.

Syntax

```
class MsHis_CodePage : MsHis_Config
{
   uint32 ID;
   string Name;
   boolean Available;
   uint32 CodePage;
   boolean DBCSEnabled;
   boolean SBCSEnabled;
   boolean MBCSEnabled;
   boolean EuroEnabled;
};
```

Properties

ID

Data Type: **uint32**

Qualifiers: **Key**

Access Type: Read-Only

The code page identifier. Used for internal reference.

Name

Data Type: **String**

Access Type: Read-Only

The name of the code page.

Available

Data Type: **Boolean**

Access Type: Read-Only

true if the code page is supported by code page translations using SNANLS; otherwise, **false**.

CodePage

Data Type: **uint32**

Access Type: Read-Only

The number of the NLS code page.

DBCSEnabled

Data Type: **Boolean**

Access Type: Read-Only

true if this code page supports Double Byte Character Sets; otherwise, **false**.

SBCSEnabled

Data Type: **Boolean**

Access Type: Read-Only

true if this code page supports Single Byte Character Sets; otherwise, **false**.

MBCSEnabled

Data Type: **Boolean**

Access Type: Read-Only

true if this code page supports Multi-Byte Character Sets; otherwise, **false**.

EuroEnabled

Data Type: **Boolean**

Access Type: Read-Only

true if this code page has Euro support; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

wmiHIS WMI Provider Classes

Administration and Management Programmer's Guide

IPC-DLC WMI Programmer's Reference

The IPC-DLC WMI Programmer's Reference describes the classes you can use to monitor the Host Integration Server 2009 link service.

In This Section

WmiSnaLinkServiceMS WMI Provider Classes

The following table describes the Common Information Model (CIM) classes used by the **WmiSnaLinkService** instance and method Windows Management Instrumentation (WMI) provider.

Class	Description
MsSna_LinkService	SNA Link service base class.
MsSna_LinkService_IpDlc	SNA IP-DLC link service.

MsSna_LinkService Class

The abstract **MsSna_LinkService** class represents an SNA link service.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LinkService
{
   String Name;
   String DllName;
   Boolean IsRemotable;
   String Title;
   String DriverName;
   uint32 Type;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key**, **MaxLen(8)** Access Type: Read-Only

A US_ASCII string containing the name of the link service. The name will be assigned automatically for a new link service.

DllName

Data Type: **String**Access Type: Read-Only

The name of the .dll that implements the link service.

IsRemotable

Data Type: **Boolean**Access Type: Read/Write

true if the link service can be used from a remote node; otherwise, **false**.

Title

Data Type: **String**Qualifiers: **MaxLen(255)** Access Type: Read/Write

The title of the link service, up to 128 symbols long.

DriverName

Data Type: **String** Access Type: Read-Only

The device driver associated with the link service.

Type

Data Type: **uint32** Access Type: Read-Only

The type of link service. The following table describes the possible values for **Type**.

Value	Meaning
3	SDLC
4	X25
10	DFT
11	DLC8022
31	TWINAX

32	CHANNEL
35	IPDLC

Requirements

Windows Server 2003, Windows XP Professional, Windows 2000

See Also

Other Resources

WmiSnaLinkServiceMS WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LinkService_IpDlc Class

The **MsSna_LinkService_IpDlc** class represents the SNA IP-DLC link service.

The following syntax is simplified from MOF code and includes all inherited properties. For reference information about methods, see the table of methods later in this topic.

Syntax

```
Class MsSna_LinkService_IpDlc : MsSNA_LinkService
{
   String Name;
   String DllName;
   Boolean IsRemotable;
   String Title;
   String DriverName;
   uint32 Type;
   String PrimaryNNS;
   String BackupNNS;
   uint32 AddressType;
   String LocalAddress;
   String NetworkName;
   String CPName;
   String NodeID;
   String LENNode;
   String ResolvedIP;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key**, **MaxLen(8)** Access Type: Read-Only

A US_ASCII string containing the name of the link service. The name will be assigned automatically for a new link service.

DllName

Data Type: **String**Access Type: Read-Only

The name of the .dll that implements the link service.

IsRemotable

Data Type: **Boolean**Access Type: Read/Write

true if the link service can be used from a remote node; otherwise, **false**. **IsRemotable** is always **false** for the IP-DLC link service.

Title

Data Type: **String**Qualifiers: **MaxLen(255)** Access Type: Read/Write

The title of the link service, up to 128 symbols long.

DriverName

Data Type: **String** Access Type: Read-Only

The device driver associated with the link service. **DriverName** is used for the IP-DLC link service.

Type

Data Type: **uint32** Access Type: Read-Only

The type of link service. The following table describes the possible values for **Type**.

Value	Meaning
3	SDLC

4	X25
10	DFT
11	DLC8022
31	TWINAX
32	CHANNEL
35	IPDLC

As an IP-DLC link service, **MsSna_LinkService_IpDlc.Type** will always be set to 35.

PrimaryNNS

Data Type: **String** Qualifiers: **MaxLen(128)** Access Type: Read/Write

Contains the name of the primary NNS server.

BackupNNS

Data Type: **String** Qualifiers: **MaxLen(1024)** Access Type: Read-Only

Contains the list of backup NNS delimited with semicolons. While **BackupNNS** may contain additional names, the current build is tested only for **BackupNNS** to be equal to the **PrimaryNNS**.

AddressType

Data Type: uint32 Access Type: Read/Write

Describes the local address type. The following table describes the possible values of **AddressType**.

Value	Description
1	ADAPTER
2	STATICIP

LocalAddress

Data Type: **String** Qualifiers: **MaxLen(256)** Access Type: Read/Write

Contains the local network adapter or address. If **AddressType** contains a 1, **LocalAddress** will contain a valid network adapter; otherwise, **LocalAddress** contains a static IP address.

NetworkName

Data Type: **String** Qualifiers: **MaxLen(8)**, **ToUpperCase** Access Type: Read/Write

The network name of the Branch Network Node as implemented by the link service. The string will be a Type A string containing up to eight symbols.

CPName

Data Type: **String** Qualifiers: **MaxLen(8)**, **ToUpperCase** Access Type: Read/Write

Control point name of the Branch Network Node as implemented by the link service. The string will be a Type A string containing up to eight symbols.

NodeID

Data Type: **String** Qualifiers: **MaxLen(9)** Access Type: Read/Write

The identity of the Branch Network Node as implemented by the link service. **NodeID** will be in format HHH.HHHHH, where H is a hexadecimal digit.

LENNode

Data Type: **String** Qualifiers: **MaxLen(20)** Access Type: Read/Write

Data Type: **String** Qualifiers: **MaxLen(20)** Access Type: Read/Write

The name of the Associated LEN node.

ResolvedIP

Data Type: **String** Access Type: Read-only

The resolved IP address of the network adapter or local address.

Methods

The following table describes the methods implemented for the **MsSna_LinkService_IpDlc** class.

Method	Description
GetAllStaticIPs	Returns the list of all the static IP addresses on the local machine.
GetAllNetworkAdapters	Returns the list of all network adapters with the IP protocols enabled.
GetNextAvailableOrdinal	Internal. Provides the next available link service number.

Requirements

Windows Server 2003, Windows XP Professional, Windows 2000

See Also

Other Resources

WmiSnaLinkServiceMS WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LinkService_IpDlc.GetAllStaticIPs Method

The **GetAllStaticIPs** method returns the list of all the static IP addresses on the local machine.

Syntax

```
void GetAllStaticIps(
   string IPs[]
);
```

Parameters

IPs

[out] Returns an array of strings containing the static IP addresses on the local machine.

Requirements

Windows Server 2003, Windows XP Professional, Windows 2000

See Also

Reference

MsSna_LinkService_IpDlc Class

Other Resources

WmiSnaLinkServiceMS WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LinkService_IpDlc.GetAllNetworkAdapters Method

The **GetAllNetworkAdapters** method returns the list of all network adapters with the IP protocols enabled.

Syntax

```
void GetAllNetworkAdapters(
   string Adapters[]
);
```

Parameters

Adapters

[out] Returns an array of strings containing the network adapters with the IP protocols enabled.

Requirements

Windows Server 2003, Windows XP Professional, Windows 2000

See Also

Reference

MsSna_LinkService_IpDlc Class

Other Resources

WmiSnaLinkServiceMS WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LinkService_IpDlc.GetNextAvailableOrdinal Method

Internal. The **GetNextAvailableOrdinal** method returns the next available link service number.

Syntax

```
uint32 GetNextAvailableOrdinal()
```

Return Value

The next available link service number.

Remarks

GetNextAvailableOrdinal is an internal method, and as such should not be used by third parties.

Requirements

Windows Server 2003, Windows XP Professional, Windows 2000

See Also

Reference

MsSna_LinkService_IpDlc Class

Other Resources

WmiSnaLinkServiceMS WMI Provider Classes

Administration and Management Programmer's Guide

SNA Trace Provider WMI Programmer’s Reference

The SNA Trace Provider WMI Programmer's Reference describes the Windows Management Instrumentation (WMI) classes you can use to capture trace messages from your Host Integration Server 2009 enterprise application.

For more information, see How to Capture a Trace with WMI.

In This Section

WmiSnaTrace WMI Provider Classes

WmiSnaTrace WMI Provider Classes

The Microsoft Host Integration Server SNA Trace provider supplies information regarding the SNA service trace. As an instance and method provider, the WmiSnaStatus provider implements the standard **IWbemProviderInit** interface and the following **IWbemServices** methods:

- **CreateInstanceEnumAsync**
- **DeleteInstanceAsync**
- **ExecMethodAsync**
- **GetObjectAsync**
- **PutInstanceAsync**

For more information on **IWbemProviderInit** and **IWbemServices**, see "COM API for WMI" in the MSDN Library at http://msdn.microsoft.com/library.

You can access these provider classes in the \root\MicrosoftHIS namespace.

Class	Description
MsHisTrace_Global	Contains the global settings for Host Integration Server tracing.
MsHisTrace_COMTI	Describes tracing properties for Transaction Integrator (TI).
MsHisTrace_SharedFoldersGateway	Contains tracing properties for the Shared Folders Gateway service.
MsHisTrace_SNAApplication	Contains tracing properties for any application that runs on top of Host Integration Server 2009.
MsHisTrace_SNAManageClient	Contains tracing properties for the SNA Manage Client.
MsHisTrace_SNAMngAgent	Contains tracing properties for the SNA Manage Agent.
MsHisTrace_SNAServerManager	Contains tracing properties for the SNA Manager.
MsHisTrace_SNABase	Contains tracing properties for the SNA Base service.
MsHisTrace_SNANetMn	Contains tracing properties for an SNA Net Manager.
MsHisTrace_SNAPrint	Contains tracing properties for the Host Print service.
MsHisTrace_SNAServer	Contains tracing properties for an SNA service
MsHisTrace_TN3270	Contains tracing properties for the TN3270 service.
MsHisTrace_Config	Contains tracing properties for the TN5250 service.
MsHisTrace_ExtendedStatus	Returns error information.
MsHisTrace_Event	Retrieves error information.

MsHisTrace_Config Class

The abstract **MsHisTrace_Config** class describes the general properties of a configuration file.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_Config
{
   string Name;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

MsHisTrace_Global Class

The **MsHisTrace_Global** class contains the global settings for Microsoft® Host Integration Server tracing.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_Global : MsHisTrace_Config
{
   string Name;
   uint32 AsyncThreadPriority;
   Boolean AsyncTraceFlag;
   uint32 FlipLength;
   string TraceFileDirectory;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

AsyncThreadPriority

Data Type: **uint32** Access Type: Read/Write

If *AsyncTraceFlag* is **true**, *AsyncThreadFlag* specifies the level of priority for tracing to run within the Windows® operating system.

AsyncTraceFlag

Data Type: **Boolean** Access Type: Read/Write

true to cause tracing to be run on a background thread; otherwise, **false**.

FlipLength

Data Type: **uint32** Access Type: Read/Write

The size a trace file should be before the trace utility switches to recording trace data in a second file.

TraceFileDirectory

Data Type: **String** Access Type: Read/Write

The directory in which to store trace files.

Requirements

Platforms: Microsoft Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_COMTI Class

The **MsHisTrace_COMTI** class describes tracing properties for Transaction Integrator (TI).

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_COMTI : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean DPLHeaderTrace;
   Boolean LU62Trace;
   Boolean BriefLU62Trace;
   Boolean COMTIProxy;
   Boolean PipeLine;
   Boolean BlackBoard;
   Boolean GeneralService;
   Boolean Repository;
   Boolean DataTransit;
   Boolean DataLayout;
   Boolean Conversions;
   Boolean Transport;
   Boolean Registrar;
   Boolean Scripting;
   Boolean SessionManager;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Qualifiers: **Qualifiers** Access Type: Read-Only

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

DPLHeaderTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of DPL Headers; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

BriefLU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable a briefer version LU 6.2 tracing; otherwise, **false**.

COMTIProxy

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the COMPI Proxy API; otherwise, **false**.

PipeLine

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the PipeLine API; otherwise, **false**.

BlackBoard

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the BlackBoard API; otherwise, **false**.

GeneralService

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the GeneralService API; otherwise, **false**.

Repository

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the Repository API; otherwise, **false**.

DataTransit

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the DataTransit API; otherwise, **false**.

Conversions

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the Conversions API; otherwise, **false**.

Transport

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the Transport API; otherwise, **false**.

Registrar

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the Registrar API; otherwise, **false**.

Scripting

Data Type: **Boolean** Access Type: Read/Write

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the Scripting API; otherwise, **false**.

SessionManager

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of the SessionManager API; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SharedFoldersGateway Class

The **MsHisTrace_SharedFoldersGateway** class contains tracing properties for the Shared Folders Gateway service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SharedFoldersGateway : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAApplication Class

The **MsHisTrace_SNAApplication** class contains tracing properties for any application that runs on top of Host Integration Server 2009.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAApplication : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap indicating which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

Requirements

Platforms: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAManageClient Class

The **MsHisTrace_SNAManageClient** contains tracing properties for the SNA Manage Client.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAManageClient : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap indicating which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAMngAgent Class

The **MsHisTrace_SNAMngAgent** class contains tracing properties for the SNA Manage Agent.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAMngAgent : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
};
```

Remarks

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap indicating which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAServerManager Class

The **MsHisTrace_SNAServerManager** class contains tracing properties for the SNA Manager.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAServerManager : MsHisTrace_Config
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap indicating which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNABase Class

The **MsHisTrace_SNABase** class contains tracing properties for the SNA Base service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNABase : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNANetMn Class

The **MsHisTrace_SNANetMn** class contains SNA Net Management trace content.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNANetMn : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean DLCTrace;
   Boolean SnaTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

DLCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable DLC tracing; otherwise, **false**.

SnaTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable SNA tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAPrint Class

The **MsHisTrace_SNAPrint** class contains tracing properties for the Host Print service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAPrint : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_SNAServer Class

The **MsHisTrace_SNAServer** class represents tracing properties for an SNA service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_SNAServer : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean DLCTrace;
   Boolean SnaTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description

1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

DLCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable DLC tracing; otherwise, **false**.

SnaTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable SNA tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_TN3270 Class

The **MsHisTrace_TN3270** class contains tracing properties for the TN3270 service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_TN3270 : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
   Boolean TraceState;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

TraceState

Data Type: **Boolean** Access Type: Read/Write

true to enable the TN3270 Internal Trace; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_Config Class

The **MsHisTrace_Config** class contains tracing properties for the TN5250 service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_TN5250 : MsHisTrace_Config
{
   string Name;
   uint32 EnabledTraces;
   Boolean InternalMessageTrace;
   Boolean T3270Trace;
   Boolean LU62Trace;
   Boolean APPCTrace;
   Boolean CPICTrace;
   Boolean LUATrace;
   Boolean CSVTrace;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Name of the configuration file.

EnabledTraces

Data Type: **uint32** Access Type: Read/Write

Bitmap that indicates which Internal Trace conditions to record. The following table describes the possible values for **EnabledTraces**.

Value	Description
1	Fatal trace mask
2	Error trace mask
4	Debug trace mask
8	State trace mask
16	Function trace mask
32	Message trace mask
64	Custom trace mask
128	Plan enter exit mask

InternalMessageTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable tracing of internal messages; otherwise, **false**.

T3270Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable T3270 tracing; otherwise, **false**.

LU62Trace

Data Type: **Boolean** Access Type: Read/Write

true to enable LU 6.2 tracing; otherwise, **false**.

APPCTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable APPC tracing; otherwise, **false**.

CPICTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CPI-C tracing; otherwise, **false**.

LUATrace

Data Type: **Boolean** Access Type: Read/Write

true to enable LUA tracing; otherwise, **false**.

CSVTrace

Data Type: **Boolean** Access Type: Read/Write

true to enable CSV tracing; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_ExtendedStatus Class

The **MsHisTrace_ExtendedStatus** class returns error information.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_ExtendedStatus : __ExtendedStatus
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

MsHisTrace_Event Class

The **MsHisTrace_Event** class retrieves error information.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisTrace_Event : __ExtrinsicEvent
{
    string Name;
    string PathToObj;
};
```

Properties

Name

Data Type: **String** Access Type: Read/Write

Name of the event that occurred.

PathToObj

Data Type: **String** Access Type: Read/Write

Path to the object that describes the event.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WmiSnaTrace WMI Provider Classes

Administration and Management Programmer's Guide

SNA Status Provider WMI Programmer's Reference

The SNA Status Provider WMI Programmer's Reference describes the Windows Management Instrumentation (WMI) classes you can use to monitor the health of your SNA servers for Host Integration Server 2009.

For more information, see How to Monitor the Health of Host Integration Server with WMI.

In This Section

WmiSnaStatus WMI Provider Classes

WmiSnaStatus WMI Provider Classes

The Microsoft® Host Integration Server SNA Status provider supplies information regarding the SNA service status. As an instance and method provider, the WmiSnaStatus provider implements the standard **IWbemProviderInit** interface and the following **IWbemServices** methods:

CreateInstanceEnumAsync

DeleteInstanceAsync

ExecMethodAsync

ExecQueryAsync

GetObjectAsync

PutInstanceAsync

For more information on **IWbemProviderInit** and **IWbemServices**, see "COM API for WMI" in the MSDN Library at http://msdn.microsoft.com/library.

The WmiSnaStatus.mof and WmiSnaStatus_XP file contains the WMISNA provider, and association and registration classes. You can access these provider classes in the \root\MicrosoftHIS namespace.

Class	Description
MsSnaStatus_EventServiceSna	Describes a change to the **EventServiceSna** class.
MsSnaStatus_EventConnection	Describes a change to the **EventConnection** class
MsSnaStatus_EventLu3270	Describes a change to the **EventLu3270** class.
MsSnaStatus_EventUser	Describes a change to the **EventUser** class.
MsSnaStatus_EventAppcLocalLu	Describes a change to the **EventAppcLocalLu** class.
MsSnaStatus_EventAppcSession	Describes a change to the **EventAppcSession** class.
MsSnaStatus_EventServicePrint	Describes a change to the **EventServicePrint** class.
MsSnaStatus_EventPrintSession	Describes a change to the **EventPrintSession** class.
MsSnaStatus_EventServiceTN3270	Describes a change to the **EventServiceTN3270** class.
MsSnaStatus_EventTN3270Session	Describes a change to the **EventTN3270Session** class.
MsSnaStatus_EventServiceTN5250	Describes a change to the **EventTN5250Session** class.
MsSnaStatus_EventServiceSharedFolder	Describes a change to the **EventServiceSharedFolder** class.
SNA_ExtendedStatus	Used to return error information if required.
MsSnaStatus_ServiceSna	Get information on the status of the SNA service.
MsSnaStatus_Connection	Represents a SNA Service connection status.
MsSnaStatus_Lu3270	Represents a SNA Service display LU, printer LU, or LUA LU status.
MsSnaStatus_ClientConnections	Represents a SNA Service client connection status.

MsSnaStatus_AppcLocalLu	Represents an APPC Local LU status.
MsSnaStatus_AppcSession	Represents an APPC session status.
MsSnaStatus_ServicePrint	Represents an Host Printer service status.
MsSnaStatus_PrintSession	Represents an Host Printer session status.
MsSnaStatus_ServiceTN3270	Represents a TN3270 service status.
MsSnaStatus_TN3270Session	Represents a TN3270 session status.
MsSnaStatus_ServiceTN5250	Represents a TN5250 service status.
MsSnaStatus_TN5250Session	Represents a TN5250 session status.
MsSnaStatus_ServiceSharedFolder	Represents a TN3270 service status.
MsSnaStatus_Lu3270ToActiveUser	Represents an association between a User connection and a 3270 type LU.
MsSnaStatus_APPCSessionToActiveUser	Represents an association between a User connection and an APPC session.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

MsSnaStatus_EventServiceSna Class

The **MsSnaStatus_EventServiceSna** class describes a change to the **EventServiceSna** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventServiceSna : MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

MsSnaStatus_EventConnection Class

The **MsSnaStatus_EventConnection** class describes a change to the **EventConnection** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventConnection: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventLu3270 Class

The **MsSnaStatus_EventLu3270** class describes a change to the **EventLu3270** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventLu3270: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventUser Class

The **MsSnaStatus_EventUser** class describes a change to the **EventUser** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventUser: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventAppcLocalLu Class

The **MsSnaStatus_EventAppcLocalLu** class describes a change to the **EventAppcLocalLu** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventAppcLocalLu: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server2003

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventAppcSession Class

The **MsSnaStatus_EventAppcSession** class describes a change to the **EventAppcSession** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventAppcSession: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventServicePrint Class

The **MsSnaStatus_EventServicePrint** class describes a change to the **EventServicePrint** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventServicePrint: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventPrintSession Class

The **MsSnaStatus_EventPrintSession** class describes a change to the **EventPrintSession** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventPrintSession: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventServiceTN3270 Class

The **MsSnaStatus_EventServiceTN3270** class describes a change to the **EventServiceTN3270** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventServiceTN3270: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventTN3270Session Class

The **MsSnaStatus_EventTN3270Session** class describes a change to the **EventTN3270Session** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventTN3270Session: MsSnaStatus_Event
{
    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventServiceTN5250 Class

The **MsSnaStatus_EventServiceTN5250** class describes a change to the **EventServiceTN5250** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventServiceTN5250: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventTN5250Session Class

The **MsSnaStatus_EventTN5250** class describes a change to the **EventTN5250Session** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventTN5250Session: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_EventServiceSharedFolder Class

The **MsSnaStatus_EventServiceSharedFolder** class describes a change to the **EventServiceSharedFolder** class.

The following syntax is simplified from MOF code.

Syntax

```
class MsSnaStatus_EventServiceSharedFolder: MsSnaStatus_Event

    object ref Path;
    sint16 Action;
};
```

Remarks

Path

Data Type: **Object ref**Access Type: Read-Only

The relative path to the corresponding object.

Action

Data Type: **sint16** Access Type: Read-Only

The event that occurred to the object specified in **Path**. The following table describes the valid values for **Action**.

Value	Description
0	Create
1	Destroy
2	Modify

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

SNA_ExtendedStatus Class

Used to return error information if required

Syntax

```
class MsSnaStatus_ExtendedStatus : __ExtendedStatus
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSna Class

The **MsSnaStatus_ServiceSna** class is used to get information on the status of the SNA service.

Syntax

```
class MsSnaStatus_ServiceSna : MsSnaStatus_Config
{
   string Name;
   sint32 Status;
   string StatusText;
   Boolean bOutOfDate;
   uint16 AppcSessions;
   uint16 LU3270Sessions;
   uint16 Users;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The name of the service. The string can contain more than one service, in the following format:<ComputerName>, <ComputerName>.01, <ComputerName>.02, <ComputerName>.03

Status

Data Type: **sint32** Access Type: Read-Only

The current status of the service. The following table describes the possible values for **Status**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values.

bOutOfDate

Data Type: **Boolean** Access Type: Read-Only

true indicates that the configuration has changed and that the service must be restarted; otherwise, **false**.

AppcSessions

Data Type:**uint16** Access Type: Read-Only

Statistics on APPC sessions on the service.

LU3270Sessions

Data Type:**uint16** Access Type: Read-Only

Statistics on LU 3270 sessions on the service.

Users

Data Type:**uint16** Access Type: Read-Only

Statistics on users on the service.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the service.
Stop	Stops the service.
Pause	Pauses the service.
Resume	Resumes the service.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Remarks

There may be more than one service running on a computer.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSna.Start Method

Starts the service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSna.Stop Method

Stops the service.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSna.Pause Method

Pauses the service.

Syntax

```
void Pause();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSna.Resume Method

Resumes the service.

Syntax

```
void Resume();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_Connection Class

The **MsSnaStatus_Connection** class represents a SNA service connection status.

The following is simplified MOF syntax.

Syntax

```
class MsSnaStatus_Connection : MsSnaStatus_Config
{
   string Name;
   string ServiceName;
   uint32 Status;
   string StatusText;
   uint32 InactiveState;
   uint16 Failcode;
   DateTime FailTime;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The name of the connection.

ServiceName

Data Type: **String** Access Type: Read-Only

Name of the SNA service to which this connection belongs.

Status

Data Type: **uint32** Access Type: Read-Only

Name of the SNA service to which this connection belongs.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values. The following table describes the potential values for **StatusText**.

Value	Description
0	
1	Stopping
2	Active

3	Incoming
4	OnDemand
5	OnDemandIncoming

InactiveState

Data Type: **uint32** Access Type: Read-Only

Additional detail on inactive connections. The following table describes the possible values for **InactiveState**.

Value	Description
0	Inactive
1	Incoming
2	OnDemand
3	OnDemandIncoming

Failcode

Data Type: **uint16** Access Type: Read-Only

The failure code. 0 indicates a non-failure.

FailTime

Data Type: **DateTime** Access Type: Read-Only

The time the failure occurred. Valid only if **Failcode** is not 0.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the connection.
Stop	Stops the connection.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_Connection.Start Method

Starts the connection.

Syntax

```
void Start();
```

Requirements

Platforms: Windows 2000, Windows XP Professional, Windows Server 2003

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_Connection.Stop Method

Stops the connection.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_Lu3270 Class

The **MsSnaStatus_Lu3270** class represents a SNA Service display LU, printer LU, or LUA LU status.

Syntax

```
class MsSnaStatus_Lu3270 : MsSnaStatus_Config
{
   string Name;
   string ConnectionName;
   uint32 Status;
   string StatusText;
   uint32 ActiveState;
   string HostAppl;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The name of the LU.

ConnectionName

Data Type: **String** Access Type: Read-Only

The connection on which this LU is defined.

Status

Data Type: **uint32** Access Type: Read-Only

The current status of the connection. The following table describes the potential values for **Status**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values. The following table describes the potential values for **StatusText**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping

4	Available
5	InSession
6	SSCP

ActiveState

Data Type: **uint32** Access Type: Read-Only

Additional detail on active ULs. The following table describes the possible values for **ActiveState**.

Value	Description
0	Available
1	InSession
2	SSCP

HostAppl

Data Type: **string** Access Type: Read-Only

The short name of the host application.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Stop	Stops the LU.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

MsSnaStatus_Lu3270.Stop Method

Stops the LU.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ClientConnections Class

The **MsSnaStatus_ClientConnections** class represents a SNA service client connection status.

Syntax

```
class MsSnaStatus_ClientConnections : MsSnaStatus_Config
{
   string UserId;
   string Domain;
   string User;
   string ClientMachine;
   DateTime ClientConnectTime;
   string ClientAppl;
};
```

Properties

UserId

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

An arbitrary unique key.

Domain

Data Type: **String** Access Type: Read-Only

The domain on which the user is located.

User

Data Type: **String** Access Type: Read-Only

The username of the user.

ClientMachine

Data Type: **String** Access Type: Read-Only

The computer name on which the client application is running.

ClientConnectTime

Data Type: **DateTime** Access Type: Read-Only

The client-server connection start time.

ClientAppl

Data Type: **string** Access Type: Read-Only

The name of the client application.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject** and **EnumerateInstances**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

MsSnaStatus_AppcLocalLu Class

The **MsSnaStatus_AppcLocalLu** class represents an APPC Local LU status.

Syntax

```
class MsSnaStatus_AppcLocalLu : MsSnaStatus_Config
{
   string Name;
   string ServiceName;
   uint16 SessionLimit;
   uint16 SessionCount;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The Local LU alias.

ServiceName

Data Type: **String** Access Type: Read-Only

The name of the SNA service on which this Local LU is defined.

SessionLimit

Data Type: **uint16** Access Type: Read-Only

The session limit for the LU.

SessionCount

Data Type: **uint16** Access Type: Read-Only

The current session count.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject** and **EnumerateInstances**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_AppcSession Class

The **MsSnaStatus_AppcSession** class represents an APPC session status.

Syntax

```
class MsSnaStatus_AppcSession : MsSnaStatus_Config
{
   string APPCLU;
   string SessionID;
   string PartnerLU;
   string Mode;
};
```

Properties

APPCLU

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The Local APPC LU.

SessionID

Data Type: **String** Access Type: Read-Only

A unique ID for this session.

PartnerLU

Data Type: **String** Access Type: Read-Only

The Partner LU alias. **PartnerLU** can be a Remote LU or another Local LU.

Mode

Data Type: **String** Access Type: Read-Only

The mode name for this session.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Stop	Stops the session.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_AppcSession.Stop Method

Stops the session.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServicePrint Class

The **MsSnaStatus_ServicePrint** class represents an SNA Print service status.

Syntax

```
class MsSnaStatus_ServicePrint : MsSnaStatus_Config
{
   string Name;
   sint32 Status;
   string StatusText;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The name of the Print Service. **Name** will be the same as the computer name.

Status

Data Type: **sint32** Access Type: Read-Only

The current status of the service. The following table describes the possible values for **Status**.

Value	Description
0	Inactive
1	Pending
2	Stopping
3	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the **Status** values.

Mode

Data Type: **String** Access Type: Read-Only

The mode name for this session.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the print service.
Stop	Stops the print service

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServicePrint.Start Method

Starts the print service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServicePrint.Stop Method

Stops the print service.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession Class

The **MsSnaStatus_PrintSession** class represents an SNA Print session status.

Syntax

```
class MsSnaStatus_PrintSession : MsSnaStatus_Config
{
   string Name;
   uint32 Status;
   string StatusText;
   uint32 PrintState;
   uint16 Type;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

A unique ID for the session.

Status

Data Type: **sint32** Access Type: Read-Only

The current status of the service. The following table describes the possible values for **Status**.

Value	Description
0	Inactive
1	Pending
2	Stopping
3	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the **Status** values. The following describes the possible values for **StatusText**.

Value
Inactive
Pending
Stopping
Spooling
Printing
Paper Out
Printer Offline
Printer Error

Printer Paused
Printer Idle
InSession
Ready
Paused
Unknown

PrintState

Data Type: **String** Access Type: Read-Only

The printer state that indicates the status of the printer or any printer errors. The following table describes the possible values for **PrintState**.

Value
Spooling
Printing
Paper Out
Printer Offline
Printer Error
Printer Paused
Printer Idle
InSession
Ready
Paused
Unknown

Type

Data Type: **uint16** Access Type: Read-Only

The type of print session. The following table describes the possible values for **Type**.

Value	**Description**
0	3270
1	APPC

Methods

Method	**Description**
GetObject	Retrieves the instance.

EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the print session.
Stop	Stops the print session.
Pause	Pauses the print session.
Restart	Restarts the print session.
PA1Key	Simulates pressing the PA1Key.
PA2Key	Simulates pressing the PA2Key.
Cancel	Cancels the print session.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.Start Method

Starts the print session.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.Stop Method

Stops the print session.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.Pause Method

Pauses the print session.

Syntax

```
void Pause();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.Restart Method

Restarts the print session.

Syntax

```
void Restart();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.PA1Key Method

Simulates pressing the PA1 key.

Syntax

```
void PA1Key();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.PA2Key Method

Simulates pressing the PA2 key.

Syntax

```
void PA2Key();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_PrintSession.Cancel Method

Cancels the print session.

Syntax

```
void Cancel();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN3270 Class

The **MsSnaStatus_ServiceTN3270** class represents a TN3270 service status.

Syntax

```
class MsSnaStatus_ServiceTN3270 : MsSnaStatus_Config
{
   string Name;
   sint32 Status;
   string StatusText;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The service name. **Name** will be identical to the local computer name.

Status

Data Type: **sint32** Access Type: Read-Only

The current status of the service. The following table describes the possible values for **Status**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the service.
Stop	Stops the service.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN3270.Start Method

Starts the service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN3270.Stop Method

Stops the service.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_TN3270Session Class

The **MsSnaStatus_TN3270Session** class represents a TN3270 session status.

Syntax

```
class MsSnaStatus_TN3270Session : MsSnaStatus_Config
{
   string Name;
   string Client;
};
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The service name. **Name** will be identical to the LUA LU or Pool name.

Client

Data Type: **String** Access Type: Read-Only

The client computer name or IP address.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN5250 Class

The **MsSnaStatus_ServiceTN5250** class represents a TN5250 service status.

Syntax

```
class MsSnaStatus_ServiceTN5250 : MsSnaStatus_Config
{
   string Name;
   sint32 Status;
   string StatusText;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The service name. **Name** will be identical to the local computer name.

Status

Data Type: **sint32** Access Type: Read-Only

The current status of the service. The following table describes the possible values for **Status**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.
ExecMethod	Executes the specified method.
Start	Starts the method.
Stop	Stops the method.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN5250.Start Method

Starts the service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceTN5250.Stop Method

Stops the service.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_TN5250Session Class

The **MsSnaStatus_TN5250Session** class represents a TN5250 session status.

Syntax

```
class MsSnaStatus_TN5250Session : MsSnaStatus_Config
{
   string TNSessionID;
   string APPCLU;
   string PartnerLU;
   string Mode;
   string Client;
};
```

Properties

TNSessionID

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

An arbitrary unique ID representing the session ID.

APPCLU

Data Type: **String** Access Type: Read-Only

APPC Local LU alias.

PartnerLU

Data Type: **String** Access Type: Read-Only

APPC Partner LU alias. **PartnerLU** can be a Remote LU or another Local LU.

Mode

Data Type: **String** Access Type: Read-Only

The mode name for this session.

Client

Data Type: **String** Access Type: Read-Only

The client computer name or IP address.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject** and **EnumerateInstances**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSharedFolder Class

The **MsSnaStatus_ServiceSharedFolder** class represents a TN3270 service status.

Syntax

```
class MsSnaStatus_ServiceSharedFolder : MsSnaStatus_Config
{
   string Name;
   sint32 Status;
   string StatusText;
}
```

Properties

Name

Data Type: **String** Qualifiers: **Key** Access Type: Read-Only

The name of the service. **Name** will be the same as the local computer name.

Status

Data Type: **sint32** Qualifiers: **Key** Access Type: Read-Only

The status of the service. The following table describes the potential values for **Status**.

Value	Description
0	
1	Inactive
2	Pending
3	Stopping
4	Active

StatusText

Data Type: **String** Access Type: Read-Only

One of the status values.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates the instance.
ExecMethod	Executes the specified method.
Start	Starts the service.
Stop	Stops the service.

For more information on **GetObject**, **EnumerateInstances**, and **ExecMethod**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

MsSnaStatus_ServiceSharedFolder.Start Method

Starts the service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_ServiceSharedFolder.Stop Method

Stops the service.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_Lu3270ToActiveUser Class

The **MsSnaStatus_Lu3270ToActiveUser** class represents an association between a User connection and a 3270 type LU.

Syntax

```
class MsSnaStatus_Lu3270ToActiveUser : MsSnaStatus_Association
{
   MsSnaStatus_Lu3270 ref PathToLU3270;
   MsSnaStatus_ActiveUser ref PathToUser;
};
```

Properties

PathToLU3270

Data Type: **MsSnaStatus_Lu3270 ref** Qualifiers: **Key** Access Type: Read-Only

The path to the APPC session.

PathToUser

Data Type: **MsSnaStatus_ActiveUser ref** Qualifiers: **Key** Access Type: Read-Only

The path to the user.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject** and **EnumerateInstances** see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsSnaStatus_APPCSessionToActiveUser Class

The **MsSnaStatus_APPCSessionToActiveUser** class represents an association between a User connection and an APPC session.

Syntax

```
class MsSnaStatus_APPCSessionToActiveUser : MsSnaStatus_Association
{
   MsSnaStatus_AppcSession ref PathToAPPCSession;
   MsSnaStatus_ActiveUser ref PathToUser;
};
```

Properties

PathToAPPCSession

Data Type: **MsSnaStatus_AppcSession ref** Qualifiers: **Key** Access Type: Read-Only

The path to the APPC session.

PathToUser

Data Type: **String**Qualifiers: **Key** Access Type: **MsSnaStatus_ActiveUser ref**

The path to the user.

Methods

Method	Description
GetObject	Retrieves the instance.
EnumerateInstances	Enumerates all instances of the object.

For more information on **GetObject** and **EnumerateInstances**, see "IWbemServices interface" in the MSDN Library at http://msdn.microsoft.com/library.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WmiSnaStatus WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

SNA Provider WMI Programmer's Reference

The SNA Provider WMI Programmer's Reference describes the Windows® Management Instrumentation (WMI) classes you can use to monitor and control SNA services.

For more information, see Controlling Services and Connections with WMI.

In This Section

- WMISNA WMI Provider Classes

WMISNA WMI Provider Classes

The Microsoft Host Integration Server SNA configuration provider supplies information regarding the SNA service configuration. As an instance and method provider, the WMISNA provider implements the standard **IWbemProviderInit** interface and the following **IWbemServices** methods:

- **CreateInstanceEnumAsync**
- **DeleteInstanceAsync**
- **ExecMethodAsync**
- **ExecQueryAsync**
- **GetObjectAsync**
- **PutInstanceAsync**

For more information on **IWbemProviderInit** and **IWbemServices**, see "COM API for WMI" in the MSDN Library at http://msdn.microsoft.com/library.

The WmiSna.mof and WmiSna_XP.mof files contain the WMISNA provider, and association and registration classes. You can access the WMISNA provider classes in the \root\MicrosoftHIS namespace.

Class	Description
MsSna_Domain	Global properties that affect all servers in the group. Sometimes called a subdoma in or OU.
MsSna_ServiceSNA	SNA service implements the SNA protocol. Contains the connections and LU reso urces.
MsSna_ServiceTN3270	TN3270 service enables clients to connect to a host via the TN3270 protocol.
MsSna_ServiceTN5250	TN5250 service enables clients to connect via the TN5250 protocol to a host.
MsSna_ServiceSharedFolder	The Shared Folders Gateway service for AS/400 file access.
MsSna_ServicePrint	Host Print service contains print sessions.
MsSna_UserInfo	Base class for a User or Group account configured in SNA.
MsSna_ConfiguredUser	A User or Group account configured in SNA. Can determine access to LUs and poo ls.
MsSna_LogInUserAndGroups	For the logged-on user, an enumeration returns the configured user and groups.
MsSna_Workstation	A workstation configured in SNA, specified by name or IP address. Can determine access to LUs and pools.
MsSna_Lu3270	Base class for 3270 LU resource. Each 3270 LU must belong to a connection.
MsSna_LuDisplay	A 3270 LU display resource. Often used for terminal emulator access.
MsSna_LuLua	A 3270 LU LUA resource. Often used for programmatic access.

MsSna_TN3270Session	The part of the TN3270 session that specifies client configuration. References a SNA_LU_LUA class.
MsSna_TN3270Port	Describes a port with security properties.
MsSna_TN3270SessionIPFilter	An IP address/name assigned to a TN3270 session. Multiple IP address/name pairs of these can be assigned to one session.
MsSna_TN5250SessionIPFilter	An IP address/name assigned to a TN5250 session. Multiple IP address/name pairs of these can be assigned to one session.
MsSna_LuPassThrough	One half of the dowstream LU/pool pairs. Represents a downstream LU/pool that is associated with a downstream connection.
MsSna_LuDown	A 3270 LU downstream resource. Reserved for use with downstream connections.
MsSna_LuPrint	A 3270 LU printer resource. Used by Print Session or by printer emulator.
MsSna_Pool	Base class for 3270 LU pools. A pool is associated with one or more 3270 LUs.
MsSna_PoolDisplay	The 3270 LU display pool.
MsSna_PoolLua	The 3270 LU LUA pool.
MsSna_PoolDown	The 3270 LU downstream pool.
MsSna_LuAppcLocal	An APPC Local LU resource. Used for LU 6.2 protocol.
MsSna_LuAppcRemote	An APPC remote LU resource. Used for LU 6.2 protocol. References a connection.
MsSna_AppcMode	An APPC Mode definition. Defines properties of an LU 6.2 session.
MsSna_Connection	Base class for SNA connection. Belongs to an SNA service. May own 3270 LUs.
MsSna_Connection8022Dlc	A type of SNA connection that uses DLC 802.2 protocol over Token Ring or Ethernet.
MsSna_ConnectionSdlc	A type of SNA connection that uses SDLC protocol over dial-up or leased lines.
MsSna_ConnectionX25	A type of SNA connection that uses X.25 protocol over dial-up or leased lines.
MsSna_ConnectionChannel	A type of SNA connection that uses Channel links.
MsSna_ConnectionDft	A type of SNA connection that uses DFT over coaxial cable.
MsSna_ConnectionTwinax	A type of SNA connection that uses twinax cable. Host Integration Server 2009 does not support twinax cable.
MsSna_Cpic	A global CPI-C definition for APPC.
MsSna_TN5250Definition	The definition of a TN5250 session.
MsSna_PrintSession	Base class for a print session on a Host Print service.

MsSna_PrintSession3270	Extends a Print session. Uses 3270 protocols to communicate with the host.
MsSna_PrintSessionAppc	Extends a Print session. Uses APPC LU 6.2 protocols to communicate with the host.
MsSna_AppcPartner	A preconfigured combination of APPC Local LU, Remote LU, and Mode.
MsSna_AccountAssigned3270	Used to query for 3270 LUs assigned to a specific workstation or user.
MsSna_AccountAssignedLua	Used to query for LUA LUs assigned to a specific workstation or user.
MsSna_AccountAssigned3270Services	Used to query for services on which a specific workstation or user has 3270 LUs/pools.
MsSna_AccountAssignedLuaServices	Used to query for services on which a specific workstation or user has LUA LUs/pools.
MsSna_AccountAvailableAppcLu	For the logged-on user account and workstation, the assigned APPC LU resources.
MsSna_AdapterOnMachine	Associates an adapter with a computer.
MsSna_ConnectionOnServer	Associates a connection with a server.
MsSna_Lu3270OnConnection	Associates a 3270 LU with a connection.
MsSna_LuDisplayAssignedToUser	Associates a display LU with a user.
MsSna_LuPrintAssignedToUser	Associates a print LU with a user.
MsSna_LuLuaAssignedToUser	Associates an LUA LU with a user.
MsSna_PoolDisplayAssignedToUser	Associates a display pool with a user.
MsSna_PoolLuaAssignedToUser	Associates the pool LUA with a user.
MsSna_LuDisplayAssignedToWorkstation	Associates a display LU with a workstation.
MsSna_LuPrintAssignedToWorkstation	Associates a print LU with a workstation.
MsSna_LuLuaAssignedToWorkstation	Associates an LUA LU with a workstation.
MsSna_PoolDisplayAssignedToWorkstation	Associates a display pool with a workstation.
MsSna_PoolLuaAssignedToWorkstation	Associates an LUA pool with a workstation.
MsSna_ConnectionUsingAdapter	Associates a connection with an adapter.
MsSna_Lu3270AssignedToPool	Associates a 3270 LU with a pool.
MsSna_PoolOnServer	Associates a pool with a server.
MsSna_ExtendedStatus	Describes the extended status of a specified message.

MsSna_Domain Class

Contains global properties that affect all servers in the group.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Domain : MsSna_Config
{
   String Name;
   String NetViewConnection;
   String DispVerbConnection;
   String EventlogServer;
   String PopupServer;
   sint16 AuditLevel;
   sint32 BroadcastMeanTime;
   boolean NamedPipes;
   boolean IpxSpx;
   boolean BanyanVines;
   boolean TcpIp;
   boolean RTMOverflow;
   boolean RTMEndOfSession;
   sint16 RTMTimerUntil;
   sint16 RTMThreshold1;
   sint16 RTMThreshold2;
   sint16 RTMThreshold3;
   sint16 RTMThreshold4;
   String ConfigServer;
   String ClientBackupSponsorNames;   // names separated by ';'
   String ClientBackupDomainName;
   sint16 ClientDomainBackupType;
   boolean Security3270;
   boolean SecurityLUA;
   boolean SecurityAPPC;
   sint32 Status;
   datetime DateTimeSaved;
};
```

Properties

Name

Data Type: **String**

Qualifiers: **Key**

Access Type: Read-Only

The name of the subdomain.

NetViewConnection

Data Type: **String**

Qualifiers: **MAXLEN(8)**

Access Type: Read/Write

A connection to which NetView data should be sent.

DispVerbConnection

Data Type: **String**

Qualifiers: **MAXLEN(8)**

Access Type: Read/Write

The default connection for Display verbs. If set to **null**, selects a random connection.

EventlogServer

Data Type: **String**

Qualifiers: **MAXLEN(16)**

Access Type: Read/Write

The name of the server on which Windows 2000 Event Logs for this server installation should be stored.

PopupServer

Data Type: **String**

Qualifiers: MAXLEN(16)

Access Type: Read/Write

The name of the server to which pop-up error messages should be routed.

AuditLevel

Data Type: **sint16**

Qualifiers: **QualifierType**

Access Type: Read/Write

The default audit log level, which determines what events, if any, are audited. The following table describes the possible values for **AuditLevel**.

Value	Description
0	Detailed
1	General
2	Significant
3	Disabled

BroadcastMeanTime

Data Type: **sint32**

Qualifiers: **MINVALUE(45), MAXVALUE(65535), UNITS("sec")** Access Type: Read/Write

The interval at which server broadcasts are repeated.

NamedPipes

Data Type: **Boolean**

Access Type: Read/Write

true to send server broadcasts over Windows Networking, also known as named pipes; otherwise, **false**.

IpxSpx

Data Type: **Boolean**

Access Type: Read/Write

true to sends server broadcasts over IPX/SPX, also known as Novell NetWare; otherwise, **false**.

BanyanVines

Data Type: **Boolean**

Access Type: Read/Write

true to send server broadcasts over Banyan VINES; otherwise, **false**.

TcpIp

Data Type: **Boolean**

Access Type: Read/Write

true to send server broadcasts over TCP/IP; otherwise, **false**.

RTMOverflow

Data Type: **Boolean**

Access Type: Read/Write

true to cause Response Time Monitor (RTM) data to be sent to the host when the number of host responses in a given time period overflows the size of the available counter; otherwise, **false**.

RTMEndOfSession

Data Type: **Boolean**

Qualifiers: **QualifierType**

Access Type: Read-Only

true to cause Response Time Monitor (RTM) data to be sent to the host at the end of each LU-to-LU session; otherwise, **false**.

RTMTimerUntil

Data Type: **sint16**

Access Type: Read-Only

The point at which the Response Time Monitor (RTM) registers that a host has responded, at which point the RTM stops the timers. The following table describes the possible values for **RTMTimerUnit**.

Value	Description
0	FirstData
1	Unlock
2	AllowSent

RTMThreshold1

Data Type: **sint16**

Qualifiers: **MINVALUE(1), MAXVALUE(1000), UNITS("tenthsec")**

Access Type: Read/Write

The cut-off times at which the RTM saves the count of host responses, and then restarts the count.

RTMThreshold2

Data Type: **sint16**

Qualifiers: **MINVALUE(1), MAXVALUE(1000), UNITS("tenthsec")**

Access Type: Read/Write

The cut-off times at which the RTM saves the count of host responses, and then restarts the count.

RTMThreshold3

Data Type: **sint16**

Qualifiers: **MINVALUE(1), MAXVALUE(1000), UNITS("tenthsec")**

Access Type: Read/Write

The cut-off times at which the RTM saves the count of host responses, and then restarts the count.

RTMThreshold4

Data Type: **sint16**

Qualifiers: **MINVALUE(1), MAXVALUE(1000), UNITS("tenthsec")**

Access Type: Read/Write

The cut-off times at which the RTM saves the count of host responses, and then restarts the count.

ConfigServer

Data Type: **sint16**

Qualifiers: **MAXLEN(16)**

Access Type: Read/Write

The primary configuration server for the subdomain.

ClientBackupSponsorNames

Data Type: **String**

Qualifiers: **MAXLEN(256)**

Access Type: Read/Write

A list describing all of the SNA sponsor servers to which the server updates the client. The names of each server must be separated with a semicolon (;).

ClientBackupDomainName

Data Type: **String**

Qualifies: **MAXLEN(16)**

Access Type: Read/Write

A list of backup SNA subdomains with which the server updates the client.

ClientDomainBackupType

Data Type: **sint16**

Access Type: Read/Write

A value describing how client backup information is sent. The following table describes the possible values for **ClientDomainBackupType**.

Value	Description
0	None
1	Domain
2	Sponsors

Security3270

Data Type: **Boolean**

Access Type: Read/Write

true if access to 3270 LUs is restricted to users assigned to the LU; otherwise, **false**.

SecurityLUA

Data Type: **Boolean**

Access Type: Read/Write

true if access to LUA LUs is restricted to users assigned to the LU; otherwise, **false**.

SecurityAPPC

Data Type: **Boolean**

Access Type: Read/Write

true if access to APPC LUs is restricted to users assigned to the LU; otherwise, **false**.

Status

Data Type: **sint32**

Access Type: Read-Only

The status of the subdomain.

DateTimeSaved

Data Type: **datetime**

Access Type: Read-Only

The time and date for which the object was saved.

Remarks

The object **MsSna_Domain** represents can also be referred to as a subdomain or OU.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

MsSna_ServiceSNA Class

Implements the SNA protocol, which contains the connections and LU resources.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ServiceSNA : MsSna_Service
{
   String Name;
   String Comment;
   String ControlPoint;
   String NetworkName;
   String StatusText;
   sint32 Status;
};
```

Properties

Name

Data Type: **String Qualifier: Key, MAXLEN(20)** Access Type: Read-Only

The service name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment field.

ControlPoint

Data Type: **String**Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The control point for this SNA node.

NetworkName

Data Type: **String**Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The Network Name name for this SNA node.

StatusText

Data Type: **String**Access Type: Read-Only

The status of the service. The following table describes the possible values for **StatusText**.

Value	Description
Not configured	The service is not currently configured.
Inactive	The service is inactive.
Stopping	The service is in the process of stopping.
Active	The service is active.
Out of Date	The service is out-of-date.
Paused	The service is paused.

Status

Data Type: **sint32**Access Type: Read-Only

The status of the service. The following table describes the status of the service:

Value	Description
0	Not configured
1	Inactive
2	Inactive
3	Stopping
4	Active
5	
6	Out-of-date
7	Paused
8	
9	

Methods

Method	Description
Start	Starts the service.
Stop	Stops the service.
Pauses	Pauses the service.
Resume	Restarts a paused service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

MsSna_ServiceSNA.Start Method

Starts the service.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceSNA.Stop Method

Stops the service.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceSNA.Pauses Method

Pauses the service.

The following syntax is simplified from MOF code.

Syntax

```
void Pause();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceSNA.Resume Method

Restarts a paused service.

The following syntax is simplified from MOF code.

Syntax

```
void Resume();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceTN3270 Class

Enables clients to connect to a host via the TN3270 protocol.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ServiceTN3270 : MsSna_Service
{
   String Name;
   String Comment;
   boolean LogNormal;
   boolean LogSna;
   boolean UseNameResolution;
   boolean PrinterFlowControl;
   boolean TNModeOnly;
   sint32 CloseDelay;
   sint32 IdleTimeout;
   sint32 InboundRU;
   sint32 OutboundRU;
   sint32 RefreshTime;
   sint32 StatusDelay;
      sint32 Port;
   String StatusText;
};
```

Properties

Name

Data Type: **String**Qualifiers: **MAXLEN(16)** Access Type: Read-Only

The TN3270 service name. **Name** is the same as the computer name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment string.

LogNormal

Data Type: **Boolean**Access Type: Read/Write

true if access to log successful client connections and successful client terminations; otherwise, **false**.

LogSna

Data Type: **Boolean**Access Type: Read/Write

true to log all TN3270 service event messages to the Event Log being used by the Host Integration Server 2009 system, instead of to the Event Log on the local computer; otherwise, **false**.

UseNameResolution

Data Type: **Boolean**Access Type: Read/Write

true to specify the name of a computer rather than the IP address—use only if you are running a domain name resolver; otherwise, **false**.

PrinterFlowControl

Data Type: **Boolean**Access Type: Read/Write

true to have the TN3270 service send all messages to a TN3270 printer client as RESPONSE-REQUIRED, and not to send any messages until it has received a response for the previous message; otherwise, **false**.

SocketListen

Data Type: **Boolean**Access Type: Read/Write

To have the TN3270 service stop listening on this socket once all of the defined LUs are in use; otherwise, **false**.

TNModeOnly

Data Type: **Boolean**Access Type: Read/Write

true to prevent the TN3270 service from using TN3270E, an enhancement to TN3270; otherwise, **false**.

CloseDelay

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(86400),UNITS("sec")** Access Type: Read/Write

The time between sending a disconnect message to the client computer and closing the socket with the client computer.

IdleTimeout

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(70560),UNITS("min")** Access Type: Read/Write

The time in which the session idles before the TN3270 service disconnects the client system. Specifying 0 means there will be no time-out.

InboundRU

Data Type: **sint32**Qualifiers: **MINVALUE(256), MAXVALUE(32768),UNITS("bytes")** Access Type: Read/Write

The RU, or SNA message, size used by the TN3270 service for logon messages from the host.

OutboundRU

Data Type: **sint32**Qualifiers: **MINVALUE(256), MAXVALUE(32768),UNITS("bytes")** Access Type: Read/Write

The RU, or SNA message, size used by the TN3270 service for logon messages to the host.

RefreshTime

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(60),UNITS("sec")** Access Type: Read/Write

The delay between updates of the status on the display.

StatusDelay

Data Type: **sint32**Qualifiers:MINVALUE**(0), MAXVALUE(86400),UNITS("sec")** Access Type: Read/Write

The delay between the time when TN3270 service connects to a host session and the time the TN3270 service starts updating the client screen.

Port

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(9999)**Access Type: Read/Write

The default port number for the TN3270 service. You can override **Port** on a per-session basis. Use 0 for the default TN3270 port.

StatusText

Data Type: **String**Access Type: Read-Only

The current status of the service. The following table describes the possible values for **StatusText**.

Value	Description
Not configured	The status is not configured.
Inactive	The status is inactive.
Pending	The status is pending.
Stopping	The status is stopping.
Active	The status is active.
Out of Date	The status is out-of-date.

Paused	The status is paused.

Methods

Method	Description
Start	Starts the service.
Stop	Stops the service.

Requirements

Platforms: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

MsSna_ServiceTN3270.Start Method

Starts the service.

The following syntax is simplified from MOF code.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceTN3270.Stop Method

Stops the service.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceTN5250 Class

Enables clients to connect via the TN5250 protocol to a host.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ServiceTN5250 : MsSna_Service
{
   String Name;
   String Comment;
   sint32 Port;
   String StatusText;
};
```

Remarks

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(16)**Access Type: Read-Only

The TN5250 service name. **Name** is the same as the computer name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

Port

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(9999)**Access Type: Read/Write

The default port number for the TN5250 service. Port can be overridden on a per-session basis. Use 0 for the default TN5250 port.

StatusText

Data Type: **String**Access Type: Read-Only

The current status of the service. The following table describes the possible values for **StatusText**.

Value	Description
Not configured	The service is not configured.
Inactive	The service is inactive.
Pending	The service is pending.
Stopping	The service is stopping.
Active	The service is active.
Out of Date	The service is out-of-date.
Paused	The service is paused.

Methods

Method	Description
Start	Starts the service.
Stop	Stops the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceTN5250.Start Method

Starts the service.

The following syntax is simplified from MOF code.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceTN5250.Stop Method

Stops the service.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceSharedFolder Class

Describes a service for AS/400 file access.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ServiceSharedFolder : MsSna_Service
{
   String Name;
   String Comment;
   String StatusText;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(16)**Access Type: Read-Only

The Shared Folder Gateway service name. **Name** is the same as the computer name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

StatusText

Data Type: **String**Access Type: Read/Write

The current status of the service. The following table describes the possible values for **StatusText**.

Value	Description
Not configured	The service is not configured.
Inactive	The service is inactive.
Pending	The service is pending.
Stopping	The service is stopping.
Active	The service is active.
Out of Date	The service is out-of-date.
Paused	The service is paused.

Methods

Method	Description
Start	Starts the service.
Stop	Stops the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

MsSna_ServiceSharedFolder.Start Method

Starts the service.

The following syntax is simplified from MOF code.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServiceSharedFolder.Stop Method

Stops the service.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServicePrint Class

Contains print sessions.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ServicePrint : MsSna_Service
{
   String Name;
   boolean NoEventLogOnSkippingTransparentSection;
   boolean UseProportionalFontChange;
   boolean FlushFinalFF;
   boolean DelayPrintStart;
   boolean AlwaysDoNL;
   boolean NoSpaceAfterFF;
   boolean DoAllFF;
   boolean IgnoreCharsUnder3F;
   boolean UseFixedTabs;
   sint32 ActivationRetryInterval;
   sint32 ActivationRetryLimit;
   String StatusText;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(16)**Access Type: Read/Write

The Host Print service name. **Name** is the same as the machine name.

NoEventLogOnSkippingTransparentSection

Data Type: **Boolean**Access Type: Read/Write

true to prevent an entry in the Event Log every time a print service skips a transparent section found while printing a host print job; otherwise, **false**.

UseProportionalFontChange

Data Type: **Boolean**Access Type: Read/Write

true to prevent overlapping characters in documents containing nonfixed type fonts printed through Host Print service; otherwise, **false**.

FlushFinalFF

Data Type: **Boolean**Access Type: Read/Write

true to cause Host Print service to explicitly form-feed the document at the end of a print job; otherwise, **false**.

DelayPrintStart

Data Type: **Boolean**Access Type: Read/Write

true to delay the start of the print job until printable data is received by Host Print services; otherwise, **false**.

AlwaysDoNL

Data Type: **Boolean**Access Type: Read/Write

true to insert a new line when the Host Print service determines that the Maximum Print Position has been reached for a particular line of data; otherwise, **false**.

NoSpaceAfterFF

Data Type: **Boolean**Access Type: Read/Write

true to prevent print services from inserting a space character following a form feed; otherwise, **false**.

DoAllFF

Data Type: **Boolean**Access Type: Read/Write

true to force the printer driver to honor all form-feed commands; otherwise, **false**.

IgnoreCharsUnder3F

Data Type: **Boolean**Access Type: Read/Write

true to cause the Host Print service to ignore hexadecimal characters 3F and below; otherwise, **false**.

UseFixedTabs

Data Type: **Boolean**Access Type: Read/Write

true to disable normal tab functionality (such as aligning tabs in columns) and interpret each tab as a fixed number of spaces; otherwise, **false**.

ActivationRetryInterval

Data Type: **sint32**Access Type: Read/Write

The number of seconds to wait before trying to print the job again. The default value is 10 seconds.

ActivationRetryLimit

Data Type: **sint32**Access Type: Read/Write

The number of times Host Print service attempts to activate the APPC conversation following a terminated connection. The default value is -1 (infinite).

StatusText

Data Type: **String**Access Type: Read/Write

The current status of Host Print service. The following table describes the possible values for **StatusText**.

Value	Description
Not configured	The service is not configured.
Inactive	The service is inactive.
Pending	The service is pending.
Stopping	The service is stopping.
Active	The service is active.
Out of Date	The service is out-of-date.
Paused	The service is paused.

Methods

Method	Description
Start	Starts the service.
Stop	Stops the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServicePrint.Start Method

Starts the service.

The following syntax is simplified from MOF code.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ServicePrint.Stop Method

Stops the service.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_UserInfo Class

Base class for a User or Group account configured in SNA.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_UserInfo : MsSna_Config
{
   String Name;
   sint32 UserType;
   boolean DynRemote;
   String DefLocalLu;
   String DefRemoteLu;
   boolean Encryption;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(255)**Access Type: Read-Only

The user name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

UserType

Data Type: **String**Access Type: Read-Only

The account type. The following table describes the possible values for **UserType**.

Value	Description
1	User
2	Group
3	Domain
4	Alias
5	WellKnownGroup
6	DeletedAccount
7	Invalid
8	Unknown
9	Computer

DynRemote

Data Type: **Boolean**Access Type: Read/Write

true to let this user or group use dynamically created APPC LUs.

DefLocalLu

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read/Write

A default local APPC LU to be used when the user starts APPC programs.

DefRemoteLu

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read/Write

A default remote APPC LU to be used when the user starts APPC programs.

Encryption

Data Type: **Boolean**Access Type: Read/Write

true to enable encryption between the client system and Host Integration Server 2009; otherwise, **false**.

Requirements

Platforms: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConfiguredUser Class

Describes a User or Group account configured in SNA.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConfiguredUser : MsSna_UserInfo
{
};
```

Remarks

MsSna_ConfiguredUser can determine access to LUs and Pools.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LogInUserAndGroups Class

Enumerates the configured user and groups for a logged-on user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LogInUserAndGroups : MsSna_UserInfo
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Workstation Class

Describes a workstation configured in SNA, specified by name or IP address.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Workstation : MsSna_Config
{
   String Name;
   sint32 IdType;
   sint16 Secure;
   String IPmask;
   boolean DynRemote;
   String Comment;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(20)**Access Type: Read-Only

The workstation ID. **Name** is usually the workstation name.

IdType

Data Type: **sint32**Access Type: Read/Write

The type of workstation specified in **Name**. The following table describes the possible values for **IdType**.

Value	Description
0	Name
1	IPAddress
2	IPSubnet

Secure

Data Type: **sint16**Access Type: Read/Write

Describes a value indicating whether the workstation can access LUs that are not directly assigned to it. The following table describes the possible values for **Secure**.

Value	Description
0	False
1	True

IPmask

Data Type: **String**Qualifiers: **MAXLEN(20)**Access Type: Read/Write

The IP Subnet mask. Use **IPmask** only if **WorkstationIdType** is set to **IPSubnet**.

DynRemote

Data Type: **Boolean**Access Type: Read/Write

true to enable users access to remote APPC LU as they are created; otherwise, **false**.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

Remarks

MsSna_Workstation can describe access to LUs and Pools.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Lu3270 Class

Describes the base class for a 3270 LU resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Lu3270 : MsSna_Resource
{
   String Name;
   String ConnectionName;
   String Comment;
   String PoolName;
   sint16 Number;
   boolean Compression;
   boolean UserWksSecure;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(8), TOUPPERCASE**Access Type: Read-Only

The LU Name.

ConnectionName

Data Type: **String**Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The connection on which this LU is defined.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

PoolName

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read-Only

The pool name. **PoolName** is valid only if the LU has already been assigned to a pool.

Number

Data Type: **sint16**Qualifiers: **MINVALUE(1), MAXVALUE(254)** Access Type: Read/Write

The LU Number for LUs on 802.2, SDLC, or X.25. For a DFT connection, **Number** is `(Port Number * 16 + LT Number)`.

Compression

Data Type: **Boolean**Access Type: Read/Write

true to enable 3270 LU data stream compression; otherwise, **false**. This option must also be set in the host VTAM configuration for the LU.

UserWksSecure

Data Type: **Boolean**Access Type: Read/Write

true to require both the user and the workstation be assigned to this LU in order to acquire it; otherwise, **false**.

Remarks

Each 3270 LU must belong to a connection.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuDisplay Class

Describes a 3270 LU display resource.

Syntax

```
class MsSna_LuDisplay : MsSna_Lu3270
{
   sint16 Model;
   boolean ModelOverride;
   String AssociatedLU;
};
```

Properties

Model

Data Type: **sint16**Access Type: Read/Write

The default display model for Terminal service. The following table describes the possible values for **Model**.

Value	Description
0	Model12
1	Model13
2	Model14
3	Model1215

ModelOverride

Data Type: **Boolean**Access Type: Read/Write

true to indicate whether the default model can be overridden; otherwise, **false**. Some emulators can only emulate certain display models.

AssociatedLU

Data Type: **String**Qualifiers:**MAXLEN(8)**Access Type: Read-Only

Associates a printer LU with the current object.

Remarks

MsSna_LuDisplay is often used for terminal emulator access.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuLua Class

Describes a 3270 LU LUA resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuLua : MsSna_Lu3270
{
   boolean HighPriorityMode;
};
```

Properties

HighPriorityMode

Data Type: **Boolean**Access Type: Read/Write

true to increase the priority for this LU; otherwise, **false**.

Remarks

MsSna_LuLua is often used for programmatic access.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_TN3270Session Class

Describes the part of the TN3270 session that specifies client configuration.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_TN3270Session : MsSna_Config
{
   String Name;
   String ConnectionName;
   String Comment;
   String PoolName;
   sint16 Number;
   boolean Compression;
   boolean UserWksSecure;
   String Service;
   sint32 TnType;
   sint32 NumSessions;
   sint32 TermTypes;
   String AssociatedLu;
   sint32 Port;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(8)**Access Type: Read-Only

The LU name.

ConnectionName

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read-Only

The connection on which this LU is defined.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

PoolName

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read-Only

If the LU has already been assigned to a pool, the pool name appears here.

Number

Data Type: **sint16**Qualifiers: **MINVALUE(1), MAXVALUE(254)** Access Type: Read/Write

The LU Number for LUs on 802.2, SDLC, or X.25. For a DFT connection, this should be the Port Number * 16 + LT Number.

Compression

Data Type: **Boolean**Access Type: Read/Write

true to enable 3270 LU data stream compression; otherwise, **false**. This option must also be set in the host VTAM configuration for the LU.

UserWksSecure

Data Type: **Boolean**Access Type: Read/Write

true to require both the user and the workstation to be assigned to this LU in order to acquire it; otherwise, **false**.

Service

Data Type: **String**Qualifiers: **Key, MAXLEN(20)**Access Type: Read/Write

The SNA service to which this LU belongs.

TnType

Data Type: **sint32**Access Type: Read/Write

The display or printer type. The following table describes the possible values for **TnType**.

Value	Description
0	Generic
1	Specific
2	GenericPrinter
3	SpecificPrinter
4	AssoicatedPrinter

NumSessions

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(65535)**Access Type: Read/Write

This is the number of TN3270 sessions allowed for the selected LU or pool.

TermTypes

Data Type: **sint32**Access Type: Read/Write

The terminal names supported by this LU. The following table describes the possible values for **TermTypes**.

Value	Description
0	3275-2
1	3276_2
2	3277_2
3	3278_2
4	3278_2_E
5	3279_2
6	3279_2_
7	E 3276_3
8	3278_3
9	3278_3_E
10	3279_3
11	3279_3_E
12	3276_4

13	3278_4
14	3278_4_E
15	3279_4
16	3279_4_E
17	3278_5
18	3278_5_E
19	3279_5
20	3279_5_E

AssociatedLu

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read/Write

Used to associate a printer LU with this display LU.

Port

Data Type: **sint32**Qualifiers: **MINVALUE(0), MAXVALUE(9999)**Access Type: Read/Write

The port to be used for this session—or 0 to use the default TN3270 port.

Remarks

MsSna_TN3270Session references a SNA_LU_LUA class.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_TN3270Port Class

Describes a port with security properties.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_TN3270Port : MsSna_Config
{
   String Service;
   sint32 Port;
   String Comment;
   sint16 Security;
   boolean ClientCertVal;
   boolean Default;
   String Name;
};
```

Properties

Service

Data Type: **String**Qualifiers: **Key, MAXLEN(20)**Access Type: Read/Write

The TN service to which this LU belongs.

Port

Data Type: **sint32**Qualifiers: **Key, MINVALUE(1), MAXVALUE(0xffff)** Access Type: Read/Write

The port number.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An option comment field.

Security

Data Type: **sint16**Access Type: Read/Write

The security level. The following table describes the possible values for **Security**.

Value	Description
0	Unsecured
1	Low
2	Medium
3	High

ClientCertVal

Data Type: **Boolean**Access Type: Read/Write

true to indicate that the client ceritificate should be validated; otherwise, **false**.

Default

Data Type: **Boolean**Qualifiers: **(TRUE)**Access Type: Read-Only

true to indicate a default port; otherwise, **false**.

Name

Data Type: **String**Qualifiers: **Something**Access Type: Read/Write

Name of the port.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_TN3270SessionIPFilter Class

An IP address or name assigned to a TN3270 session.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_TN3270SessionIPFilter : MsSna_Config
{
   String Name;
   String Session;
   sint16 Type;
   String SubnetMask;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(15)**Access Type: Read-Only

The IP address or name of the computer assigned to the TN3270 session.

Session

Data Type: **String**Qualifiers: **Key, MAXLEN(8)**Access Type: Read-Only

The TN3270 session name.

Type

Data Type: **sint16**Qualifiers: **Key**Access Type: Read/Write

A value that determines if **Name** contains an IP address or a name. The following table describes the possible values for **Type**.

Value	Description
0	Name
1	IPAddress

SubnetMask

Data Type: **String**Access Type: Read/Write

The IP Subnet mask, if an IP address is specified for **Name**.

Remarks

Multiple IP addresses or names can be assigned to one session.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_TN5250SessionIPFilter Class

Contains the IP address or name assigned to a TN5250 session.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_TN5250SessionIPFilter : MsSna_Config
{
   String Name;
   String AS400;
   sint16 Type;
   String SubnetMask;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(15)**Access Type: Read-Only

The IP address or name of the computer assigned to the TN5250 session.

AS400

Data Type: **String**Qualifiers: **Key, MAXLEN(8)**Access Type: Read-Only

The AS/400 definition.

Type

Data Type: **sint16**Qualifiers: **Key**A value that indicates whether **Name** contains an IP address or a name. The following table describes the possible values for **Type**.

Value	Description
0	Name
1	IPAddress

SubnetMask

Data Type: **String**Access Type: Read/Write

The IP Subnet mask, if an IP address is specified for 'Name'.

Remarks

You may assign multiple IP addresses or names in a single session.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuPassThrough Class

Contains one half of a dowstream LU/pool pair.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuPassThrough : MsSna_Config
{
   String Connection;
   sint16 Number;
   String Name;
   boolean IsPool;
};
```

Properties

Connection

Data Type: **String**Qualifiers: **Key, MAXLEN(8)**Access Type: Read-Only

The associated downstream connection.

Number

Data Type: **sint16**Qualifiers: **Key, MINVALUE(0)**Access Type: Read-Only

The number of the downstream connection.

Name

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read/Write

The downstream LU or pool name.

IsPool

Data Type: **Boolean**Access Type: Read/Write

true to indicate that the object is a downstream a pool; **false** to indicate that the object is a downstream LU.

Remarks

MsSna_LuPassThrough represents a downstream LU/pool associated with a downstream connection.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuDown Class

A 3270 LU downstream resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuDown   : MsSna_Lu3270
{
};
```

Remarks

MsSna_LuDown is reserved for use with downstream connections.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuPrint Class

Describes a 3270 LU printer resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuPrint   : MsSna_Lu3270
{
   String AssociatedLU;
};
```

Properties

AssociatedLU

Data Type: **String**Qualifiers: **MAXLEN(8)**Access Type: Read/Write

The associated display LU.

Remarks

MsSna_LuPrint is used by Print session or by printer emulation.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Pool Class

Base class for 3270 LU pools.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Pool : MsSna_Resource
{
   String Name;
   String Comment;
};
```

Properties

Name

Data Type: **String**Qualifiers: **Key, MAXLEN(8)**Access Type: Read-Only

The pool name.

Comment

Data Type: **String**Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

Remarks

A pool is associated with one or more 3270 LUs.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolDisplay Class

Contains the 3270 LU Display pool.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolDisplay : MsSna_Pool
{
   sint16 Model;
   boolean ModelOverride;
   boolean AssocPrint;
};
```

Properties

Model

Data Type: **sint16** Access Type: Read/Write

The default display model for terminal emulation. The following table describes the possible values for **Model**.

Value	Description
0	Model2
1	Model3
2	Model4
3	Model5

ModelOverride

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the default model can be overridden; otherwise, **false**. Some emulators can only emulate certain display models.

AssocPrint

Data Type: **Boolean** Qualifiers: **Something** Access Type: Read/Write

true to associate a printer LU with this display LU pool; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolLua Class

The 3270 LU LUA pool.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolLua : MsSna_Pool
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolDown Class

The 3270 LU downstream pool.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolDown : MsSna_Pool
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuAppcLocal Class

An APPC local LU resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuAppcLocal : MsSna_Config
{
   String Service;
   String Alias;
   String Comment;
   String NetName;
   String LUName;
   boolean PoolMember;
   String IncomingLu;
   sint16 TpTimeout;
   sint16 Number;
   string Connection;
   boolean SyncPoint;
   String SyncPointClient;
};
```

Properties

Service

Data Type: **String** Qualifiers: **Key, MAXLEN(20), TOUPPERCASE** Access Type: Read-Only

The SNA service to which this LU belongs.

Alias

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read-Only

The LU alias.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment field.

NetName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The network name, which must match that configured on the remote computer.

LUName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The LU name, which can be the same as the LU alias.

PoolMember

Data Type: **Boolean** Access Type: Read/Write

true to indicate that this LU is a member of the default outgoing Local APPC LU pool; otherwise, **false**.

IncomingLu

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

An implicit incoming remote LU.

TpTimeout

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(3600)** Access Type: Read/Write

The time-out value, in seconds. If you want to manually start the invokable TP, be sure to specify at least 60 seconds; this will give the operator sufficient time to perform the necessary actions.

Number

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(254)** Access Type: Read/Write

The LU number, for dependent LUs.

Connection

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The connection on which a dependent LU is configured.

SyncPoint

Data Type: **Boolean** Access Type: Read/Write

true to indicate that you have a very specialized transaction program (TP) that requires Resync Service; otherwise, **false**.

SyncPointClient

Data Type: **String** Qualifiers: **MAXLEN(15)** Access Type: Read/Write

The IP address or the name of the client computer.

Remarks

MsSna_LuAppcLocal is used for LU 6.2 protocols.

Requirements

Platforms: Microsoft Windows Server 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuAppcRemote Class

An APPC remote LU resource.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuAppcRemote : MsSna_Config
{
   String Service;
   String Connection;
   String Alias;
   String Comment;
   String NetName;
   String LUName;
   String UnName;
   boolean Parallel;
   String IncomingMode;
   sint16 Security;
   String SecKeyHex;
   String SecKeyChar;
   boolean SyncPoint;
};
```

Properties

Service

Data Type: **String** Qualifiers: **MAXLEN(20), TOUPPERCASE** Access Type: Read-Only

The SNA service to which this LU belongs.

Connection

Data Type: **String** Qualifiers: **Key,MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The connection on which a dependent LU is configured.

Alias

Data Type: **String** Qualifiers: **Key, MAXLEN(8)** Access Type: Read-Only

The LU Alias.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment field.

NetName

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The network name. You can obtain the correct name from the host or peer administrator.

LUName

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The LU name, which can be the same as the LU alias.

UnName

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The uninterpreted LU name for LUs, which are paired with a dependent local APPC LU.

Parallel

Data Type: **String** Access Type: Read/Write

A value that indicates that parallel sessions will be used. **Parallel** must be used with an independent, local LU.

IncomingMode

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The implicit incoming mode for this LU.

Security

Data Type: **sint16** Access Type: Read/Write

A value that indicates what sort of session-level security will be used with this LU. The following table describes the possible values for **Security**.

Value	Description
0	None
1	Hex
2	Characters

SecKeyHex

Data Type: **String** Qualifiers: **MAXLEN(16)** Access Type: Read/Write

The security key, in hexadecimal. Requires that **Security** be set to '**1**'.

SecKeyChar

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The security key, in characters. Requires that **Security** be set to **2**.

SyncPoint

Data Type: **Boolean** Access Type: Read/Write

true to enable **SyncPoint**; otherwise, **false**. **SyncPoint** requires that the Local LU alias be unique.

Remarks

MsSna_LuAppcRemote is used for LU 6.2 protocol, and references a connection.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AppcMode Class

Contains an APPC Mode definition.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AppcMode : MsSna_Config
{
   String Name;
   String Comment;
   sint32 SessionLimit;
   sint32 MinContWin;
   sint32 PartMinContWin;
   sint32 AutoActivate;
   boolean Priority;
   sint32 TxPacing;
   sint32 RxPacing;
   sint32 TxRu;
   sint32 RxRu;
   sint16 MaxSendCompression;
   sint16 MaxRcvCompression;
   boolean AllowLZandRLE;
   boolean EndpointOnly;
};
```

Properties

Name

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read-Only

The mode name.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment field.

SessionLimit

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(15000)** Access Type: Read/Write

The maximum number of parallel sessions allowed with this mode.

MinContWin

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(15000)** Access Type: Read/Write

The minimum contention winner limit.

PartMinContWin

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(15000)** Access Type: Read/Write

The partner minimum contention winner limit.

AutoActivate

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(15000)** Access Type: Read/Write

The number of contention winner sessions to be activated for the local LU whenever the connection for this mode is started.

Priority

Data Type: **Boolean** Access Type: Read/Write

true to give preference to communication with this mode over communication with a low-priority mode; otherwise, **false**.

TxPacing

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(63)** Access Type: Read/Write

The maximum number of frames for the local LU to send without an SNA pacing response from the partner LU. Setting **TxPacing** to 0 indicates a unlimited number of frames.

RxPacing

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(63)** Access Type: Read/Write

The maximum number of frames for the local LU to receive from the partner LU before the local LU sends an SNA pacing response. Setting **RxPacing** to **0** (zero) indicates a unlimited number of frames.

TxRu

Data Type: **sint32** Qualifiers: **MINVALUE(256), MAXVALUE(65527)** Access Type: Read/Write

The maximum size for RUs sent by the TPs on the local system.

RxRu

Data Type: **sint32** Qualifiers: **MINVALUE(256), MAXVALUE(65527)** Access Type: Read/Write

The maximum size for RUs received from the TPs on the remote system.

MaxSendCompression

Data Type: **sint32** Access Type: Read/Write

The maximum compression allowed when sending. The following table describes the possible values for **MaxSendCompression**.

Values	Description
0	None
1	RLE
2	LZ9

MaxRcvCompression

Data Type: **sint16** Access Type: Read/Write

The maximum compression allowed when receiving. The following table describes the possible values for **MaxRcvCompression**.

Values	Description
0	None
1	RLE
2	LZ9

AllowLZandRLE

Data Type: **Boolean** Access Type: Read/Write

true to compress data using RLE before being further compressed using LZ9; otherwise, **false**. **AllowLZandRLE** is valid only if you are using LZ9.

EndpointOnly

Data Type: **Boolean** Access Type: Read/Write

true to allow intermediate nodes to use compression if one of the endpoints does not support or otherwise does not use compression; otherwise, **false**.

Remarks

MsSna_AppcMode defines properties of an LU 6.2 session.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Connection Class

Base class for SNA connection.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Connection : MsSna_Config
{
   String Service;
   String Name;
   String Comment;
   String Adapter;
   sint32 RemoteEnd;
   sint16 PeerRole;
   sint16 Activation;
   boolean AllowIncoming;
   boolean DynamicLuDef;
   String PartnerConnectionName;
   String BlockId;
   String NodeId;
   String RemoteNetName;
   String RemoteControlPoint;
   String RemoteBlockId;
   String RemoteNodeId;
   sint16 XIDFormat;
   String LocalNetName;
   String LocalControlPoint;
   sint16 CompressionLevel;
   sint16 RetryLimit;
   sint16 RetryDelay;
   String StatusText;
}
```

Properties

Service

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read-Only

The SNA service to which this connection belongs.

Name

Data Type: **String** Qualifiers: **Key, MAXLEN(8)** Access Type: Read-ONly

The connection name.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)** Access Type: Read/Write

An optional comment field.

Adapter

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The link service to be used by this connection.

RemoteEnd

Data Type: **sint32** Access Type: Read/Write

The remote end. The following table describes the possible values of **RemoteEnd**.

Value	Description
0	Host

1	Peer
2	Downstream
3	Passthrough

For 3270 or LUA access, be sure to set **RemoteEnd** to *Host*.

PeerRole

Data Type: **String** Access Type: Read/Write

The peer role. The following table describes the possible values of **PeerRole**.

Value	Description
0	Primary
1	Secondary
2	Negotiable
-1	Invalid

PeerRole only applies to connections with **RemoteEnd** set to **Peer** or **Passthrough**.

Activation

Data Type: **sint16** Access Type: Read/Write

The activation setting. Applicable only if Outgoing Calls are included in Allowed Directions. The following table describes the possible values for **Activation**.

Value	Description
0	Initial
1	OnDemand
2	Administrator
3	Incoming

AllowIncoming

Data Type: **Boolean** Access Type: Read/Write

true to indicate that incoming calls are allowed; otherwise, **false**.

DynamicLuDef

Data Type: **Boolean** Access Type: Read/Write

true to automatically configure the APPC Remote LUs as users request them; otherwise, **false**. **DynamicLuDef** requires that an APPN End Node or Net Node be available on the connection.

PartnerConnectionName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The name of the connecting partner. Valid only if **RemoteEnd** is of **Passthrough**.

BlockId

Data Type: **String** Qualifiers: **MAXLEN(3), TOUPPERCASE** Access Type: Read/Write

The block ID.

NodeId

Data Type: **String** Qualifiers: **MAXLEN(5), TOUPPERCASE** Access Type: Read/Write

The local node ID. Applicable only for connections which use a switched SDLC line (standard telephone line) to connect to a host system.

RemoteNetName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The network name of the remote system. Applicable if you are using Format 3 XIDs.

RemoteControlPoint

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The control point of the reomote system. Applicable if you are using Format 3 XIDs.

RemoteBlockId

Data Type: **String** Qualifiers: **MAXLEN(3), TOUPPERCASE** Access Type: Read/Write

The remote block ID.

RemoteNodeId

Data Type: **String** Qualifiers: **MAXLEN(5), TOUPPERCASE** Access Type: Read/Write

The remote node ID.

XIDFormat

Data Type: **sint16** Access Type: Read/Write

The XID Type. The following table describes the possible values for **XIDFormat**.

Value	Description
0	Format 0
3	Format 3

Most systems use Format 3.

LocalNetName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The network name that, along with the **LocalControlPoint**, identifies a system.

LocalControlPoint

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The control point that, along with **LocalNetName**, identifies a system.

CompressionLevel

Data Type: **sint16** Access Type: Read/Write

The compression type used by this connection. The following table describes the possible values for **CompressionLevel**.

Value	Description
0	None
1	RLE
2	LZ9

3	LZ10
4	LZ12

RetryLimit

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(255)** Access Type: Read/Write

The number of times the local system tries to send data to the remote system if there is no response.

RetryDelay

Data Type: **String** Qualifiers: **MINVALUE(0), MAXVALUE(255), UNITS("sec")** Access Type: Read/Write

The length of time for the local system to wait for a response to a transmission before trying again.

StatusText

Data Type: **String** Access Type: Read-Only

The current status of the connection. The following table describes the possible values of **StatusText**.

Value	Description
0	Unconfigured
1	Inactive
2	Incoming
3	OnDemand
4	OnDemand/Incoming
5	Pending
6	Stopping
7	Active
8	Error
9	

Name

Data Type: **String** Qualifiers: **Something** Access Type: Read/Write

The name of the connection.

Methods

Method	Description
Start	Starts the connection.
Stop	Stops the connection.
ExchangePassthroughLus	Returns a value that determines if two LUs are part of a passthrough connection pair.

Remarks

MsSna_Connection belongs to a SNA service, and may own 3270 LUs.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Connection.Start Method

Starts the connection.

The following syntax is simplified from MOF code.

Syntax

```
void Start();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Connection.Stop Method

Stops the connection.

The following syntax is simplified from MOF code.

Syntax

```
void Stop();
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Connection.ExchangePassthroughLus Method

Returns a value that determines if two LUs are part of a passthrough connection pair.

The following syntax is simplified from MOF code.

Syntax

```
boolean ExchangePassthroughLus
(
```

Parameters

FirstLu

The first LU.

SecondLu

The second LU.

Return Value

true if the LUs are part of a passthrough pair; otherwise, **false**.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Connection8022Dlc Class

Describes a type of SNA connection that uses DLC 802.2 protocol over a Token Ring or Ethernet.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Connection8022Dlc : MsSna_Connection
{
   String Address;
   sint16 DlcRetryLimit;
   sint16 DlcTimerT1;
   sint16 DlcTimerT2;
   sint16 DlcXidRetry;
   sint16 DlcRecvThresh;
   sint16 DlcSendLimit;
   sint16 DlcRemoteSAP;
   sint16 DLCType;
   sint16 DlcLocalSAP;
   sint32 MaxBtu;
};
```

Parameters

Address

Data Type: **String** Qualifiers: **MAXLEN(12), TOUPPERCASE** Access Type: Read/Write

The 12-digit hexadecimal remote network address. You can receive the correct value for **Address** by contacting the administrator of the remote system.

DlcRetryLimit

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(255)** Access Type: Read/Write

The number of times that the local system should retransmit a frame if no response is received from the remote system.

DlcTimerT1

Data Type: **sint16** Access Type: Read/Write

The amount of time that the local system should wait for the remote system to respond to a transmission before the local system tries again. The following table describes the possible values for **DlcTimer**.

Value	Description
0	Default
1	200 ms
2	400 ms
3	600 ms
4	800 ms
5	1000 ms
6	1 s
7	2 s
8	3 s

9	4 s
10	5 s

DlcTimerT2

Data Type: **sint16** Access Type: Read/Write

The maximum amount of time that should be allowed before the local system sends an acknowledgment of a received transmission. The following table describes the possible values for **DlcTimerT2**.

Value	Description
0	Default
1	40 ms
2	80 ms
3	120 ms
4	160 ms
5	200 ms
6	400 ms
7	800 ms
8	1200 ms
9	1600 ms
10	2000 ms

DlcXidRetry

Data Type: **sint16** Qualifiers: **QualiferValueHere** Access Type: Read/Write

The amount of time that the link can be inactive before the local system treats it as nonfunctioning and shuts it down. The following table describes the possible values for **DlcXidRetry**.

Value	Description
0	1 s
1	2 s
2	3 s
3	4 s
4	5 s
5	5 s (2)
6	10 s

7	15 s
8	20 s
9	25 s

DlcXidRetry

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(30)** Access Type: Read/Write

The number of times that the local system should retransmit an XID, an identifying message, if no response is received from the remote system.

DlcRecvThresh

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(127)** Access Type: Read/Write

The maximum number of frames that the local system can receive from the remote system before sending a response.

DlcSendLimit

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(127)** Access Type: Read/Write

The maximum number of frames that the local system can send without receiving a response from the remote system.

DlcRemoteSAP

Data Type: **sint16** Qualifiers: **MINVALUE(4), MAXVALUE(252)** Access Type: Read/Write

A 2-digit hexadecimal number that is a multiple of 04, between 04 and EC. See your Token Ring or Ethernet manual for more information.

DLCType

Data Type: **sint16** Access Type: Read-Only

The DLC type. The following table describes the possible values for **DLCType**.

Value	Description
0	Token
1	Ether

DlcLocalSAP

Data Type: sint16 Qualifiers: **MINVALUE(4), MAXVALUE(252)** Access Type: Read/Write

A 2-digit hexadecimal number that is a multiple of 04, between 04 and EC. See your Token Ring or Ethernet manual for more information.

MaxBtu

Data Type: **sint16** Qualifiers: **MINVALUE(265), MAXVALUE(16393)** Access Type: Read/Write

The maximum Basic Transmission Unit (BTU) Length.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionSdlc Class

A type of SNA connection that uses SDLC protocol over dial-up or leased lines.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionSdlc : MsSna_Connection
{
   String DialData;
   sint16 SdlcEncoding;
   sint16 SdlcDuplex;
   String SdlcPollAddress;
   boolean SdlcLeasedLine;
   sint16 SdlcDataRate;
   sint16 SdlcContactLimit;
   sint16 SdlcContactTO;
   sint16 SdlcIdleLimit;
   sint16 SdlcIdleTO;
   boolean SdlcMultiPrimary;
   sint16 SdlcPollLimit;
   sint16 SdlcPollTO;
   sint16 SdlcPollRate;
   boolean SdlcStandby;
   sint16 SdlcSwitchTO;
   sint32 MaxBtu;
};
```

Parameters

DialData

Data Type: **String** Qualifiers: **MAXLEN(40)** The dial data for connections which use a switched line and the phone number is not stored in the modem.

SdlcEncoding

Data Type: **sint16** Access Type: Read/Write

The encoding scheme (NRZ or NRZI) for the modem to use. The following table describes the possible values for **3dlcEncoding**.

Value	Description
0	NRZ
1	NRZI

SdlcEncoding must match the remote modem.

SdlcDuplex

Data Type: **sint16** Access Type: Read/Write

The duplex setting of the modem. The following table describes the possible values for **SdlcDuplex**.

Value	Description
0	Half
1	Full

SdlcPollAddress

Data Type: **String** Qualifiers: **MAXLEN(2)** Access Type: Read/Write

A 2-digit hexadecimal number describing the poll address. Contact the administrator of the remote system to determine the appropriate value.

SdlcLeasedLine

Data Type: **Boolean** Access Type: Read-Only

The value depends on the Link service.

SdlcDataRate

Data Type: **sint16** Access Type: Read/Write

The data rate for transmissions between the communications adapter and the modem. The following table describes the possible values for **SdlcDataRate**.

Value	Description
0	Low
1	High

SdlcContactLimit

Data Type: **String** Qualifiers: **MINVALUE(1), MAXVALUE(20)** Access Type: Read/Write

The maximum number of times the link service resends an XID or SNRM before declaring an outage to Host Integration Server 2009.

SdlcContactTO

Data Type: **String** Qualifiers: **MINVALUE(5), MAXVALUE(300)** Access Type: Read/Write

The length of time, in tenths of a second, which the SDLC link service waits for an XID or SNRM response before resending the XID or SNRM.

SdlcIdleLimit

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(255)** Access Type: Read/Write

The number of times for the local system to try to send data to the remote system if there is no response.

SdlcIdleTO

Data Type: **sint16** Access Type: **MINVALUE(1), MAXVALUE(300)**

The length of time, in tenths of a second, for the local system to wait for a response to a transmission before trying again.

SdlcMultiPrimary

Data Type: **Boolean** Access Type: Read/Write

true to indicate this is a leased SDLC line to downstream systems, and this server will be the primary station for a multidrop connection; otherwise, **false**.

SdlcPollLimit

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(255)** Access Type: Read/Write

The number of times for the local system to poll the remote system, if there is no response.

SdlcPollTO

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(300)** Access Type: Read/Write

The length of time, in tenths of a second, for the local system to wait for a response to a poll before trying again.

SdlcPollRate

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(50)** Access Type: Read/Write

The poll rate, in polls per second. Valid only if the remote system is a peer or downstream system.

SdlcStandby

Data Type: **Boolean** Access Type: Read/Write

true to set the Standby line to of a modem; otherwise, **false**.

SdlcSwitchTO

Data Type: **sint16** Qualifiers: **MINVALUE(10), MAXVALUE(500)** Access Type: Read/Write

The number of seconds that are allowed for the user or modem to dial and make a connection to the remote computer.

MaxBtu

Data Type: **sint32** Qualifiers: **MINVALUE(265), MAXVALUE(16393)** Access Type: Read/Write

The Maximum Basic Transmission Unit (BTU) length.

Requirements

Platforms: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionX25 Class

Describes a type of SNA connection that uses X.25 protocol over dial-up or leased lines.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionX25 : MsSna_Connection
{
   String Address;
   sint16 PVCAlias;
   sint16 VCType;
   String X25Facility;
   String X25UserData;
   sint16 X25PacketSize;
   sint16 X25WindowSize;
   sint32 MaxBtu;
};
```

Parameters

Address

Data Type: **String** Qualifiers: **MAXLEN(15), TOUPPERCASE** Access Type: Read/Write

The Switched Virtual Circuit (SVC) address of the host for connections using switched virtual circuit.

PVCAlias

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(65535)** Access Type: Read/Write

The Permanent Virtual Circuit (PVC) alias of the host for connections using permanent virtual circuit.

VCType

Data Type: **sint16** Access Type: Read/Write

The type of virtual circuit used by the connection. The following table describes the possible values for **VCType**.

Value	Description
0	Switched
1	Permanent

X25Facility

Data Type: **String** Qualifiers: **MAXLEN(132), TOUPPERCASE** Access Type: Read/Write

The codes for any Facility Data required by your network provider or by the administrator of the remote system. Valid only for an SVC channel.

X25UserData

Data Type: **String** Qualifiers: **MAXLEN(32), TOUPPERCASE** Access Type: Read/Write

The codes for any User Data required by your network provider. Valid only for an SVC channel.

X25PacketSize

Data Type: **sint16** Qualifiers: **QualiferValueHere** Access Type: Read/Write

The maximum number of data bytes, not header bytes, to be sent in a frame. The following table describes the possible values for **X25PacketSize**.

Value	Description
0	64

1	128
2	256
3	512
4	1024

X23PacketSize is valid only for a PVC channel.

X25WindowSize

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(7)** Access Type: Read/Write

The maximum number of frames that the local system can send without receiving a response from the remote system. Valid only for a PVC channel.

MaxBtu

Data Type: **sin32** Qualifiers: **MINVALUE(265), MAXVALUE(16393)** Access Type: Read/Write

The Maximum Basic Transmission Unit (BTU) length.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionChannel Class

A type of SNA connection that uses channel links.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionChannel : MsSna_Connection
{
   String Address;
   String CtrlUnit;
   sint32 MaxBtu;
};
```

Properties

Address

Data Type: **String** Qualifiers: **MAXLEN(2)** Access Type: Read/Write

A 2-digit hexadecimal number identifying the channel. The range is from 00 through FF; the default is FF.

CtrlUnit

Data Type: **String** Qualifiers: **MAXLEN(1)** Access Type: Read-Only

A value for the control unit image number. The range is 0 through F; the default is 0 (zero).

MaxBtu

Data Type: **String** Qualifiers: **MINVALUE(265), MAXVALUE(65536)** Access Type: Read/Write

The Maximum Basic Transmission Unit (BTU) length.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionDft Class

A type of SNA connection that uses DFT over a coaxial cable.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionDft : MsSna_Connection
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionTwinax Class

A type of SNA connection that uses a Twinax cable.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionTwinax : MsSna_Connection
{
};
```

Remarks

Twinax is not supported by Host Integration Server 2009.

Requirements

Operating Systems: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Cpic Class

A global CPI-C definition for APPC.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Cpic : MsSna_Config
{
   String Name;
   String ApplicationTPName;
   String ServiceTPHexName;
   String FullLUName;
   String FullNetName;
   sint16 SecurityType;
   String UserId;
   String Password;
   String AliasLUName;
   sint16 LUNameType;
   sint16 TPNameType;
   String ModeName;
};
```

Parameters

Name

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read-Only

The Symbolic Destination name.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

ApplicationTPName

Data Type: **String** Qualifiers: **MAXLEN(64)**Access Type: Read/Write

The application TP name.

ServiceTPHexName

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

A hexadecimal value describing the SNA service TP number.

FullLUName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The partner LU name. Valid only for identifying a partner LU with a fully qualified name.

FullNetName

Data Type: **String** Qualifiers: **MAXLEN(8)**Access Type: Read/Write

The partner network name. Valid only for identifying a partner LU with a fully qualified name.

SecurityType

Data Type: **sint16** Qualifiers: **QualiferValueHere** Access Type: Read/Write

The **Conversation Security** option to be used. The following table describes the possible values for **SecurityType**.

Value	Description
0	None

1	Same
2	Program

UserId

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The user ID to be used when specifying *Program* for **SecurityType**.

Password

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The password to be used when specifying *Program* for **SecurityType**

AliasLUName

Data Type: **String** Qualifiers: MAXLEN(8), TOUPPERCASE Access Type: Read/Write

An alias. Valid only when identifying the Partner LU by alias.

LUNameType

Data Type: **sint16** Access Type: Read/Write

A value that indicates whether the Partner LU name is specified with an alias or a fully qualified LU name. The following table describes the possible values for **LUNameType**.

Value	Description
0	Alias
1	Full

TPNameType

Data Type: **sint16** Qualifiers: **QualiferValueHere** Access Type: Read/Write

A value that indicates whether the Partner TP name is specified in characters or hex. The following table describes the possible values for **TPNameType**.

Value	Description
0	Character
1	Hex

ModeName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The name of the mode to be used.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_TN5250Definition Class

The definition of a TN5250 session.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_TN5250Definition : MsSna_Config
{
   String Name;
   String Service;
   String RemoteLUAlias;
   String LocalLUAlias;
   String Comment;
   String User;
   String Password;
   String Mode;
   sint32 TermTypes;
   sint32 Port;
};
```

Parameters

Name

Data Type: **String** Qualifiers**: Key,MAXLEN(10)** Access Type: Read/Write

The session name.

Service

Data Type: **String** Qualifiers: **MAXLEN(20), TOUPPERCASE** Access Type: Read/Write

The SNA service to which this session belongs.

RemoteLUAlias

Data Type: **String** Qualifiers: **MAXLEN(8)**Access Type: Read/Write

The local LU alias used in the session.

LocalLUAlias

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The local LU alias used in the session.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

User

Data Type: **String** Qualifiers: **MAXLEN(10)**Access Type: Read/Write

The AS/400 user name used in the session.

Password

Data Type: **String** Qualifiers: **MAXLEN(10)**Access Type: Read/Write

The AS/400 password used in the session.

Mode

Data Type: **String** Access Type: Read/Write

The mode used in the session (QPCSUPP).

TermTypes

Data Type: **String** Qualifiers: **QualiferValueHere** Access Type: Read/Write

The terminal types allowed for the session. The following list describes the possible values for **TermType.**

- 5555_C01
- 5555_B01
- 3477_FC
- 3180_2
- 3179_2
- 3196_A1
- 5292_2
- 5291_1
- 5251_11

Port

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(9999)**Access Type: Read/Write

The port used for the session. By default the value is 0.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PrintSession Class

A base class for a print session on a Print service.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PrintSession : MsSna_Config
{
   String Name;
   String Service;
   String Comment;
   String StatusText;
   sint16 Activation;
   sint16 CodePage;
   sint16 CodePageLanguage;
   String PrinterDeviceName;
   String CodePageCustomFile;
   String PrinterFile;
   boolean PrintToFile;
   String FaceName;
   boolean FaceNameOverride;
   sint32 LeftMargin;
   sint32 RightMargin;
   sint32 TopMargin;
   sint32 BottomMargin;
   boolean MarginOverride;
   boolean UniqueExtension;
   String PDTFile;
   boolean CheckPDTFile;
   String Filter;
   boolean bFilter;
   sint16 FontSize;
   sint16 SessionType;
   sint16 LinesPerInch;
   sint16 CharsPerLine;
   boolean IgnoreTransparentSections;
   boolean NoHorizontalScaling;
   boolean NoVerticalScaling;
   boolean LPIOverride;
   boolean PageSetupOverride;
};
```

Parameters

Name

Data Type: **String** Qualifiers: **Key, MAXLEN(32), TOUPPERCASE**Access Type: Read/Write

The session name, which distinguishes different printers on the network.

Service

Data Type: **String** Qualifiers: **MAXLEN(20)**Access Type: Read/Write

The SNA service to which the print session belongs.

Comment

Data Type: **String** Qualifiers: **MAXLEN(25)**Access Type: Read/Write

An optional comment field.

StatusText

Data Type: **String** Access Type: Read/Write

The status of the print session.

Activation

Data Type: **String** Access Type: Read/Write

The print session activation. The following table describes the possible values for **Activation**.

Value	Description
0	Automatic. Activates the print session automatically when the Host Print service is started
1	Manual. Activates the print session manually.

CodePage

Data Type: **sint16**Access Type: Read/Write

A value that indicates whether a standard language code or a custom code page will be used. The following table describes the possible values for **CodePage**.

Value	Description
0	Language
1	Custom

CodePageLanguage

Data Type: **sint16**Access Type: Read/Write

The code page to be used in the print session. For more information about the possible values for **CodePageLanguage**, see the **Remarks** section.

PrinterDeviceName

Data Type: **String** Qualifiers: **MAXLEN(256)**Access Type: Read/Write

The name of the destination printer.

CodePageCustomFile

Data Type: **String** Qualifiers: **MAXLEN(256)**Access Type: Read/Write

The file name if a custom code page is to be used.

PrinterFile

Data Type: **String** Qualifiers: **MAXLEN(256)**Access Type: Read/Write

The name of the file. Valid only when printing to a file.

PrintToFile

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the print job will be sent to a file; otherwise, **false**. Note that you must still configure a destination printer.

FaceName

Data Type: **String** Qualifiers: **MAXLEN(31)**Access Type: Read/Write

The name of the face.

FaceNameOverride

Data Type: **Boolean** Access Type: Read/Write

true to override the host commands; otherwise, **false**.

LeftMargin

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(255)**Access Type: Read/Write

The left margin, in inches.

RightMargin

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(255)** Access Type: Read/Write

The right margin, in inches.

TopMargin

Data Type: **String** Qualifiers: **MINVALUE(0), MAXVALUE(255)** Access Type: Read/Write

The top margin, in inches.

BottomMargin

Data Type: **sint32** Qualifiers: **MINVALUE(0), MAXVALUE(255)**Access Type: Read/Write

The bottom margin, in inches.

MarginOverride

Data Type: **Boolean** Access Type: Read/Write

true to override the host margin commands; otherwise, **false**.

UniqueExtension

Data Type: **Boolean** Access Type: Read/Write

true to instruct the print service to give each file a unique extension when printing a file.

PDTFile

Data Type: **String** Qualifiers: **MAXLEN(256)**Access Type: Read/Write

A PDT file used to format the print job.

CheckPDTFile

Data Type: **Boolean** Access Type: Read/Write

true to indicate that a PDT file will be used to format the print job; otherwise, **false**.

Filter

Data Type: **String** Qualifiers: **MAXLEN(256)**Access Type: Read/Write

The filter DLL to be used to filter the printer data stream.

bFilter

Data Type: **Boolean** Qualifiers: **QualiferValueHere** Access Type: Read/Write

true to indicate that a filter DLL will be used to filter the printer data stream; otherwise, **false**.

FontSizeOverride

Data Type: **Boolean** Access Type: Read/Write

true to override the host font size commands.

FontSize

Data Type: **sint16** Qualifiers: **MINVALUE(0), MAXVALUE(3276)** Access Type: Read/Write

The font size to be used when printing.

SessionType

Data Type: **sint16** Access Type: Read/Write

A value that indicates whether this is an APPC or 3270 print session. The following table describes the possible values for **SessionType**.

Value	Description
0	APPC

1	3270

LinesPerInch

Data Type: **sint16** Qualifiers: **MINVALUE(1), MAXVALUE(12)** Access Type: Read/Write

The number of lines per inch to be printed.

CharsPerLine

Data Type: **sint16** Access Type: Read/Write

The number of characters per line to be printed.

IgnoreTransparentSections

Data Type: **Boolean** Access Type: Read/Write

true to ignore sections of the print data stream that have been marked as Transparent; otherwise, **false**. This value is valid only when using a PDT file to format the data.

NoHorizontalScaling

Data Type: **Boolean** Access Type: Read/Write

true to turn off off the horizontal scaling feature of the printer driver; otherwise, **false**.

NoVerticalScaling

Data Type: **Boolean** Access Type: Read/Write

true to turn off off the vertical scaling feature of the printer driver; otherwise, **false**.

LPIOverride

Data Type: **Boolean** Access Type: Read/Write

true to enable host commands for lines per inch to be overridden; otherwise, **false**.

PageSetupOverride

Data Type: **String** Access Type: Read/Write

The override for the page setup.

Remarks

The following table describes the possible values for **CodePageLanguage**.

0	Afrikaans[500]
1	Albanian[870]
2	Arabic (Algeria)[420]
3	Arabic (Kingdom of Bahrain)[420]
4	Arabic (Egypt)[420]
5	Arabic (Iraq)[420]
6	Arabic (Jordan)[420]
7	Arabic (Kuwait)[420]
8	Arabic (Lebanon)[420]

9	Arabic (Libya)[420]
10	Arabic (Morocco)[420]
11	Arabic (Oman)[420]
12	Arabic (Qatar)[420]
13	Arabic (Saudi Arabia)[420]
14	Arabic (Syria)[420]
15	Arabic (Tunisia)[420]
16	Arabic (U.A.E.)[420]
17	Arabic (Yemen)[420]
18	Basque[284]
19	Belarusian[1025]
20	Bulgarian[1025]
21	Catalan[284]
22	Chinese (PRC)[935]
23	Chinese (Singapore)[935]
24	Chinese (Hong Kong)[937]
25	Chinese (Macau)[937]
26	Chinese (Taiwan)[937]
27	Croatian[870]
28	Czech[870]
29	Danish[277]
30	Dutch (Belgium)[500]
31	Dutch (Standard)[037]
32	English (Australian)[037]
33	English (Belize)[500]
34	English (Canadian)[037]
35	English (Caribbean)[500]

36	English (Ireland)[285]
37	English (Jamaica)[500]
38	English (New Zealand)[037]
39	English (South Africa)[037]
40	English (Trinidad)[500]
41	English (United Kingdom)[285]
42	English (United States)[037]
43	Estonian[1112]
44	Faeroese[277]
45	Finnish[278]
46	French (Belgium)[500]
47	French (Canadian)[037]
48	French (Luxembourg)[500]
49	French (Standard)[297]
50	French (Swiss)[500]
51	German (Austrian)[273]
52	German (Liechtenstein)[500]
53	German (Luxembourg)[500]
54	German (Standard)[273]
55	German (Swiss)[500]
56	Greek[423]
57	Greek (Modern)[875]
58	Hebrew[424]
59	Hungarian[870]
60	Icelandic[871]
61	Indonesian[037]
62	Italian[280]

63	Italian (Swiss)[500]
64	International[500]
65	Japanese (Extend Katakana)[930]
66	Japanese (English-lower)[931]
67	Japanese (Extend English)[939]
68	Japanese (Katakana)[290]
69	Korean[933]
70	Latvian[1112]
71	Lithuanian[1112]
72	Macedonian[1025]
73	Malay[037]
74	Norwegian (Bokmal)[277]
75	Norwegian (Nynorsk)[277]
76	Polish[870]
77	Portuguese (Brazil)[037]
78	Portuguese (Portugal)[037]
79	Romanian[870]
80	Russian[880]
81	Serbian (Cyrillic)[1025]
82	Serbian (Latin)[870]
83	Slovak[870]
84	Slovenian[870]
85	Spanish (Argentina)[284]
86	Spanish (Bolivia)[284]
87	Spanish (Chile)[284]
88	Spanish (Columbia)[284]
89	Spanish (Costa Rica)[284]

90	Spanish (Dominican Rep.)[284]
91	Spanish (Ecuador)[284]
92	Spanish (El Salvador)[284]
93	Spanish (Guatemala)[284]
94	Spanish (Honduras)[284]
95	Spanish (Mexico)[284]
96	Spanish (Modern Sort)[284]
97	Spanish (Nicaragua)[284]
98	Spanish (Panama)[284]
99	Spanish (Paraguay)[284]
100	Spanish (Peru)[284]
101	Spanish (Puerto Rico)[284]
102	Spanish (Trad. Sort)[284]
103	Spanish (Uruguay)[284]
104	Spanish (Venezuela)[284]
105	Swedish[278]
106	Thai[838]
107	Turkish[905]
108	Turkish (Latin-5)[1026]
109	Ukrainian[1025]
110	Danish (Euro)[1142]
111	English (Canadian) (Euro)[1140]
112	English (United Kingdom) (Euro)[1146]
113	English (United States) (Euro)[1140]
114	Finnish (Euro)[1143]
115	French (Standard) (Euro)[1147]
116	German (Standard) (Euro)[1141]

117	Icelandic (Euro)[1149]
118	International (Euro)[1148]
119	Italian (Euro)[1144]
120	Latin-1 Open System (Euro)[924]
121	Norwegian (Bokmal) (Euro)[1142]
122	Norwegian (Nynorsk) (Euro)[1142]
123	Spanish (Trad. Sort) (Euro)[1145]
124	Swedish (Euro)[1143]
125	Latin-1 Open System[1047]
126	English (Australian) (Euro)[1140]
127	French (Canadian) (Euro)[1140]

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PrintSession3270 Class

Extends a print session.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PrintSession3270 : MsSna_PrintSession
{
   sint16 JobTermination;
   sint16 JobTimeoutSecs;
   boolean JobTimeout;
   boolean UseGDI;
   boolean TRNisASCII;
   boolean CustomTRN;
   sint32 CustomTRNChar;
   String LUName;
   boolean MonitorJob;
   boolean LineWrap;
   boolean JobRenderedAtHost;
};
```

Parameters

JobTermination

Data Type: **sint16** Access Type: Read/Write

'EndBracket' means the job completes when the print server receives the End Bracket notification—otherwise, the print server spools the job until the session ends. The following table describes the possible values for **JobTermination**.

Value	Description
0	EndBracket
1	UnBind

JobTimeoutSecs

Data Type: **sint16** Qualifiers: **MINVALUE(0),MAXVALUE(99),UNITS("sec")** Access Type: Read/Write

The timeout for terminating 3270 print jobs—use with 'JobTimeout'.

JobTimeout

Data Type: **Boolean** Access Type: Read/Write

true to indicate whether a time-out will be used; otherwise, **false**. **JobTimeout** is most often used with **JobTimeoutSecs**.

UseGDI

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the Windows Graphical Device Interface (GDI) is used to format the print job; otherwise, **false**.

TRNisASCII

Data Type: **Boolean** Access Type: Read/Write

true to indicate that transparent data from the host is in ASCII and needs no translation from EBCDIC to ASCII; otherwise, **false**.

CustomTRN

Data Type: **Boolean** Access Type: Read/Write

true to indicate that a custom transparency byte, such as one other than the IBM standard of 0x35, will be used; otherwise, **false**.

CustomTRNChar

Data Type: **sint32** Qualifiers: **MINVALUE(0),MAXVALUE(255)**Access Type: Read/Write

The custom transparency byte—use this with 'CustomTRN' set to **true**.

LUName

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The 3270 printer LU for the print session.

MonitorJob

Data Type: **Boolean** Access Type: Read/Write

true to cause the print server to send a message to the host stating that the print job completed; otherwise, **false**.

LineWrap

Data Type: **Boolean** Access Type: Read/Write

true to line wrap; otherwise, **false**.

JobRenderedAtHost

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the print job is rendered at the host rather than by the Host Integration Server Host Print service; otherwise, **false**.

Remarks

MsSna_PrintSession3270 uses 3270 protocols to communicate with the host.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PrintSessionAppc Class

Extends a print session. Uses APPC LU 6.2 protocols to communicate with the host.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PrintSessionAppc : MsSna_PrintSession
{
   String LocalLUAlias;
   String RemoteLUAlias;
   String RemoteFQName;
   sint16 Remote;
   String Mode;
   String UserId;
   String Password;
   String AS400Device;
   sint16 System36;
   sint16 HostPrintTransform;
   sint16 DeviceType;
   String FontId;
   String MsgQName;
   String MsgLibName;
   String PrinterName;
   sint16 PaperSrc1;
   sint16 PaperSrc2;
   sint16 CodePage899;
   String SpecialObjName;
   String SpecialLibName;
};
```

Parameters

LocalLUAlias

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The local LU alias to be used for this print session. Valid only when a **RemoteLUAlias** is provided.

RemoteLUAlias

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The remote LU alias. Valid only if a fully qualified name is not provided.

RemoteFQName

Data Type: **String** Qualifiers: **MAXLEN(17)**Access Type: Read/Write

The remote LU fully qualified name. Valid only if **RemoteLUAlias** and **LocalLUAlias** are not provided.

Remote

Data Type: **sint16** Access Type: Read/Write

A value that specifies what type of connection to use: either a remote APPC LU alias or a fully qualified name. The following table describes the possible values for **Remote**.

Value	Description
0	Alias
1	Fully qualified name

Mode

Data Type: **String** Qualifiers: **MAXLEN(8), TOUPPERCASE**Access Type: Read/Write

The name of the mode to use. The default mode name is QPCSUPP.

UserId

Data Type: **String** Qualifiers: **MAXLEN(10)**Access Type: Read/Write

The remote user name.

Password

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The remote password.

AS400Device

Data Type: **String** Qualifiers: **MAXLEN(10), TOUPPERCASE** Access Type: Read/Write

The name for the AS/400 printer device. **AS400Device** should be a descriptive name that distinguishes different printers on the network.

System36

Data Type: **sint16** Access Type: Read/Write

A value that indicates whether the remote system is an AS/400 or a System/36. The following table describes the possible values for **System36**.

Value	Description
0	AS400
1	System/36

HostPrintTransform

Data Type: **sint16** Access Type: Read/Write

A value that indicates whether Host Print Transform (HPT) will be used. The following table describes the possible values for **HostPrintTransform**.

Value	Description
0	FALSE
1	TRUE

DeviceType

Data Type: **sint16** Access Type: Read/Write

The print device for Host Print Transform. The following table describes the possible values for **DeviceType**.

Value	Description
0	5224
1	3812

The default value for **DeviceType** is 5224.

FontId

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

A font, which should be used instead of the default.

MsgQName

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The qualified name of the message queue to which operational messages for this device are sent.

MsgLibName

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The name of the library in which the message queue is located.

PrinterName

Data Type: **TOUPPERCASE** Access Type: Read/Write

The printer type to be used with Host Print Transform. For more information on the possible values for **PrinterName**, see the **Remarks** section.

PaperSrc1

Data Type: **sint16** Access Type: Read/Write

The type of paper used in paper source 1. The following table describes the possible values for **PaperSrc1**.

Value	Description
0	Default
1	Letter (8.5 x 11 inches)
2	Legal (8.5 x 14 inches)
3	Executive (7.25 x 10.5 inches)
4	A4 (210 x 297 mm)
5	A5 (148 x 210 mm)
6	B5 (182 x 257 mm)
7	Continuous Form (8.0 inches)
8	Continuous Form (13.2 inches)
9	None

PaperSrc2

Data Type: **sint16** Access Type: Read/Write

The type of paper used in paper source 2. The following table describes the possible values for **PaperSrc2**.

Value	Description
0	Default
1	Letter (8.5 x 11 inches)
2	Legal (8.5 x 14 inches)
3	Executive (7.25 x 10.5 inches)
4	A4 (210 x 297 mm)
5	A5 (148 x 210 mm)
6	B5 (182 x 257 mm)

7	Continuous Form (8.0 inches)
8	Continuous Form (13.2 inches)
9	None

CodePage899

Data Type: **sint16** Access Type: Read/Write

A value that indicates whether code page 899 is used. The following table describes the possible values of **CodePage899**.

Value	Description
0	FALSE
1	TRUE

SpecialObjName

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The name of the special object.

SpecialLibName

Data Type: **String** Qualifiers: **MAXLEN(10)** Access Type: Read/Write

The name of the special library.

Remarks

The following table describes the possible values for **PrinterName**.

Value	Description
0	*IBM2380
1	*IBM2381
2	*IBM2390
3	*IBM2391
4	*IBM3812
5	*IBM3816
6	*IBM3912HP
7	*IBM3916HP
8	*IBM39302
9	*IBM39303
10	*IBM4019
11	*IBM4019HP
12	*IBM4029

13	*IBM4029HP
14	*IBM4037
15	*IBM4037HP
16	*IBM4070
17	*IBM4070EP
18	*IBM42011
19	*IBM42012
20	*IBM42013
21	*IBM42021
22	*IBM42022
23	*IBM42023
24	*IBM42071
25	*IBM42072
26	*IBM42081
27	*IBM4212
28	*IBM4216
29	*IBM4226
30	*IBM4230
31	*IBM4232
32	*IBM47121
33	*IBM47122
34	*IBM47221
35	*IBM47222
36	*IBM4770
37	*IBM5152
38	*IBM5210
39	*IBM5202

40	*IBM5204
41	*IBM5216
42	*IBM6408
43	*IBM6412
44	*CPQPM15
45	*CPQPM20
46	*HPII
47	*HPIID
48	*HPIIP
49	*HPIIID
50	*HPIIIP
51	*HPIIISI
52	*HP4
53	*HP500
54	*HP550C
55	*HP560C
56	*HPPAINT
57	*EPAP2250
58	*EPAP3250
59	*EPAP5000
60	*EPAP5500
61	*EPDFX5000
62	*EPDFX8000
63	*EPDFX850
64	*EPDFX870
65	*EPDFX1170

66	*EPLQ570
67	*EPLQ860
68	*EPLQ870
69	*EPLQ1070
70	*EPLQ1170
71	*EPLX810
72	*EPLQ510
73	*EPLQ2550
74	*EPSQ870
75	*EPSQ1170
76	*EPEPL7000
77	*EPEPL8000
78	*OKI320IBM
79	*OKI321IBM
80	*OKI390IBM
81	*OKI393IBM
82	*OKI590IBM
83	*OKI591IBM
84	*OKI400
85	*OKI800
86	*OKI810
87	*OKI820
88	*OKI3410
89	*NECP2
90	*NEC2200
91	*NECP2200XE
92	*NECP5200

93	*NECP5300
94	*NECP6200
95	*NECP6300
96	*PAN1123EP
97	*PAN1124
98	*PAN1124IEP
99	*PAN1180EP
100	*PAN1180IEP
101	*PAN1191
102	*PAN1624EP
103	*PAN1654EP
104	*PAN1695EP
105	*PAN2123EP
106	*PAN2124
107	*PAN2180
108	*PAN2624EP
109	*PAN4410
110	*PAN4420
111	*PAN4430
112	*PAN4450IHP
113	*PAN4451HP
114	*XRX4215MRP
115	*XRX4219MRP
116	*XRX4220MRP
117	*XRX4235
118	*XRX4700II
119	*WSCST

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AppcPartner Class

A preconfigured combination of APPC local LU, remote LU, and Mode.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AppcPartner : MsSna_Config
{
   String Mode;
   String Server;
   String LocalLUAlias;
   String PartnerLUAlias;
   String Connection;
};
```

Properties

Mode

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The mode to be used in the partnership.

Server

Data Type: **String** Qualifiers: **Key, MAXLEN(16), TOUPPERCASE** Access Type: Read/Write

The Host Integration Server name on which this partnership will be added.

LocalLUAlias

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The Local LU alias to be used in the partnership.

PartnerLUAlias

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The partner LU. **PartnerLUAlias** can be a Remote LU or a different Local LU.

Connection

Data Type: **String** Qualifiers: **Key, MAXLEN(8), TOUPPERCASE** Access Type: Read/Write

The connection name for the Remote LU.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AccountAssigned3270 Class

This is used to query for 3270 LUs assigned to a specific workstation or user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AccountAssigned3270 : MsSna_Assigned
{
   String Wks;
   String IPAddress;
   String MacAddress;
   String Account;
   String Service;
   String LU;
   boolean IsPool;
   boolean IsAssociatedPrinterLU;
   boolean ModelOverridable;
   sint16 Model;
};
```

Parameters

Wks

Data Type: **String** Qualifiers**: MAXLEN(20)**Access Type: Read/Write

The workstation name. By default, returns LUs assigned to all workstations.

IPAddress

Data Type: **String** Qualifiers: **MAXLEN(20)**Access Type: Read/Write

The workstation IP. By default, returns LUs assigned to all workstations.

MacAddress

Data Type: **String** Qualifiers: **MAXLEN(20)**Access Type: Read/Write

A value that returns LUs configured to a specific media access control (MAC) address.

Account

Data Type: **String** Access Type: Read/Write

The user for which to query for LUs. By default, will return LUs assigned to all users.

Service

Data Type: **String** Access Type: Read/Write

The SNA service on which to query for LUs.

LU

Data Type: **String** Qualifiers: Key, MAXLEN(8) Access Type: Read/Write

A value that indicates whether the returned LU is part of an LU pool.

IsPool

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the returned LU is part of an LU pool; otherwise, **false**.

IsAssociatedPrinterLU

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the returned LU has an Associated Printer LU; otherwise, **false**.

ModelOverridable

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the returned LU has an display model which can be overridden; otherwise, **false**.

Model

Data Type: **sint16** Access Type: Read/Write

The default display model of the returned LU. The following table describes the possible values for **Model**.

Value	Description
0	Model12
1	Model13
2	Model14
3	Model15

Remarks

MsSna_AccountAssigned3270 is used in querying for LUA LUs assigned to a workstation as well as to the specified user.

On specifying "" for **Workstation**, only the LUs for the user will be returned.

On specifying "*" for **User**, the LUs for the logged in user will be returned.

If User is omitted, LUs assigned to all Users will be returned.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AccountAssignedLua Class

Used to query for LUA LUs assigned to a specific workstation or user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AccountAssignedLua : MsSna_Assigned
{
   String Wks;
   String IPAddress;
   String MacAddress;
   String Account;
   String Service;
   String LU;
   boolean IsPool;
   boolean ModelOverridable;
};
```

Properties

Wks

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation name. By default, LUs assigned to all workstations are returned.

IPAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation IP. By default, LUs assigned to all workstations are returned.

MacAddress

Data Type: **String** Access Type: Read/Write

The user for which to query for LUs. By default, returns LUs assigned to all users.

Account

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

The name of the service.

Service

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

The SNA service on which to query for LUs.

LU

Data Type: **String** Qualifiers: **Key, MAXLEN(8)** Access Type: Read/Write

The name of the LU.

IsPool

Data Type: **Boolean** Access Type: Read/Write

true to indicate that the returned LU is part of an LU pool; otherwise, **false**.

ModelOverridable

Data Type: **Boolean** Access Type: Read/Write

true if the LU has a display model which can be overridden.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AccountAssigned3270Services Class

Used to query for services on which a specific workstation or user has 3270 LUs/pools.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AccountAssigned3270Services : MsSna_Assigned
{
   String Wks;
   String IPAddress;
   String MacAddress;
   String Account;
   String Service;
};
```

Properties

Wks

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation name. By default, those assigned to all workstations are returned.

IPAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read-Only

The workstation IP address. By default, those assigned to all workstations are returned.

MacAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

Returns services with LUs configured to a specific media access control (MAC) address.

Account

Data Type: **String** Access Type: Read/Write

The user to which the LUs are assigned. By default, returns services with LUs assigned to all users.

Service

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

The name of the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AccountAssignedLuaServices Class

Used to query for services on which a specific workstation or user has LUA LUs/pools.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AccountAssignedLuaServices : MsSna_Assigned
{
   String Wks;
   String IPAddress;
   String MacAddress;
   String Account;
   String Service;
};
```

Properties

Wks

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation name. By default, those assigned to all workstations are returned.

IPAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation IP address. By default, those assigned to all workstations are returned.

MacAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

Returns services with LUs configured to a specific media access control (MAC) address.

Account

Data Type: **String** Access Type: Read/Write

The user to which the LUs are assigned. By default, returns services with LUs assigned to all users.

Service

Data Type: **String** Qualifiers: **Key** Access Type: Read/Write

Returns the name of the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AccountAvailableAppcLu Class

For the logged-on user account and workstation, the assigned APPC LU resources.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AccountAvailableAppcLu : MsSna_Assigned
{
   String LocalLu;
   String RemoteLu;
   String Wks;
   String IPAddress;
   String MacAddress;
};
```

Properties

LocalLu

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The local LU name.

RemoteLu

Data Type: **String** Qualifiers: **MAXLEN(8)** Access Type: Read/Write

The remote LU name.

Wks

Data Type: **String** Qualifiers: **MAXLEN(20** Access Type: Read/Write

The workstation name.

IPAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The workstation IP address.

MacAddress

Data Type: **String** Qualifiers: **MAXLEN(20)** Access Type: Read/Write

The MAC address.

Remarks

MsSna_AccountAvailableAppcLu is used in querying for the remote APPC LUs for a given local APPC LU on a given workstation and associated with the logged-on user.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_AdapterOnMachine Class

Associates an adapter with a computer.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_AdapterOnMachine : MsSna_Association
{
   MsSna_Adapter ref PathToAdapter;
   MsSna_Server ref PathToServer;
};
```

Properties

MsSna_Adapter

Data Type: **ref PathToAdapter** Qualifiers: **Key** Access Type: Read-Only

The adapter to associate with.

MsSna_Server

Data Type: **ref PathToServer** Qualifiers: **Key** Access Type: Read-Only

The server to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionOnServer Class

Associates a connection with a server.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionOnServer : MsSna_Association
{
   MsSna_Connection ref PathToConnection;
   MsSna_Server ref PathToServer;
};
```

Properties

MsSna_Connection

Data Type: **ref PathToConnection** Qualifiers: **Key** Access Type: Read-Only

The connection to associate with.

MsSna_Server

Data Type: **ref PathToServer** Qualifiers: **Key** Access Type: Read-Only

The path to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Lu3270OnConnection Class

Associates a 3270 LU with a connection.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Lu3270OnConnection : MsSna_Association
{
   MsSna_Connection ref PathToConnection;
   MsSna_Lu3270 ref PathToLu3270;
};
```

Properties

MsSna_Connection

Data Type: **ref PathToConnection** Qualifiers: **Key** Access Type: Read-Only

The connection to associate with.

MsSna_Lu3270

Data Type: **ref PathToLu3270** Qualifiers: **Key** Access Type: Read-Only

The 3270 LU to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuDisplayAssignedToUser Class

Associates a display LU with a user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuDisplayAssignedToUser : MsSna_Lu3270AssignedToUser
{
   MsSna_LuDisplay ref PathToLuDisplay;
   MsSna_ConfiguredUser ref PathToUser;
};
```

Properties

MsSna_LuDisplay

Data Type: **ref PathToLuDisplay** Qualifiers: **Key** Access Type: Read-Only

The display LU to associate with.

MsSna_ConfiguredUser

Data Type: **ref PathToUser** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuPrintAssignedToUser Class

Associates a print LU with a user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuPrintAssignedToUser : MsSna_Lu3270AssignedToUser
{
   MsSna_LuPrint ref PathToLuPrint;
   MsSna_ConfiguredUser ref PathToUser;
};
```

Properties

MsSna_LuPrint

Data Type: **ref PathToLuPrint** Qualifiers: **Key** Access Type: Read-Only

The print LU to associate with.

MsSna_ConfiguredUser

Data Type: **ref PathToUser** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuLuaAssignedToUser Class

Associates an LUA LU with a user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuLuaAssignedToUser : MsSna_Lu3270AssignedToUser
{
   MsSna_LuLua ref PathToLuLua;
   MsSna_ConfiguredUser ref PathToUser;
};
```

Properties

MsSna_LuLua

Data Type: **ref PathToLuLua** Qualifiers: **Key** Access Type: Read-Only

The LUA LU to associate with.

MsSna_ConfiguredUser

Data Type: **ref PathToUser** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolDisplayAssignedToUser Class

Associates a display pool with a user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolDisplayAssignedToUser : MsSna_PoolAssignedToUser
{
   MsSna_PoolDisplay ref PathToPoolDisplay;
   MsSna_ConfiguredUser ref PathToUser;
};
```

Properties

MsSna_PoolDisplay

Data Type: **ref PathToPoolDisplay** Qualifiers: **Key** Access Type: Read-Only

The display pool to associate with.

MsSna_ConfiguredUser

Data Type: **ref PathToUser** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolLuaAssignedToUser Class

Associates a pool LUA with a user.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolLuaAssignedToUser : MsSna_PoolAssignedToUser
{
   MsSna_PoolLua ref PathToPoolLua;
   MsSna_ConfiguredUser ref PathToUser;
};
```

Properties

MsSna_PoolLua

Data Type: **ref PathToPoolLua** Qualifiers: **Key** Access Type: Read-Only

The pool LUA to associate with.

MsSna_ConfiguredUser

Data Type: **ref PathToUser** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuDisplayAssignedToWorkstation Class

Associates a display LU with a workstation.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuDisplayAssignedToWorkstation : MsSna_Lu3270AssignedToWorkstation
{
   MsSna_LuDisplay ref PathToLuDisplay;
   MsSna_Workstation ref PathToWorkstation;
};
```

Properties

MsSna_LuDisplay

Data Type: **ref PathToLuDisplay** Qualifiers: **Key** Access Type: Read-Only

The display LU to associate with.

MsSna_Workstation

Data Type: **ref PathToWorkstation** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuPrintAssignedToWorkstation Class

Associates a print LU with a workstation.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuPrintAssignedToWorkstation : MsSna_Lu3270AssignedToWorkstation
{
   MsSna_LuPrint ref PathToLuPrint;
   MsSna_Workstation ref PathToWorkstation;
};
```

Properties

MsSna_LuPrint

Data Type: **ref PathToLuPrint** Qualifiers: **Key** Access Type: Read-Only

The print LU to associate with.

MsSna_Workstation

Data Type: **ref PathToWorkstation** Qualifiers: **Key** Access Type: Read-Only

The user to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_LuLuaAssignedToWorkstation Class

Associates an LUA LU with a workstation.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_LuLuaAssignedToWorkstation : MsSna_Lu3270AssignedToWorkstation
{
   MsSna_LuLua ref PathToLuLua;
   MsSna_Workstation ref PathToWorkstation;
};
```

Properties

MsSna_LuLua

Data Type: **ref PathToLuLua** Qualifiers: **Key** Access Type: Read-Only

The LUA LU to associate with.

MsSna_Workstation

Data Type: **ref PathToWorkstation** Qualifiers: **Key** Access Type: Read-Only

The workstation to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolDisplayAssignedToWorkstation Class

Associates a display pool with a workstation.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolDisplayAssignedToWorkstation : MsSna_PoolAssignedToWorkstation
{
   MsSna_PoolDisplay ref PathToPoolDisplay;
   MsSna_Workstation ref PathToWorkstation;
};
```

Properties

MsSna_PoolDisplay

Data Type: **ref PathToPoolDisplay** Qualifiers: **Key** Access Type: Read-Only

The display pool to associate with.

MsSna_Workstation

Data Type: **ref PathToWorkstation** Qualifiers: **Key** Access Type: Read-Only

The workstation to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolLuaAssignedToWorkstation Class

Associates an LUA pool with a workstation.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolLuaAssignedToWorkstation : MsSna_PoolAssignedToWorkstation
{
   MsSna_PoolLua ref PathToPoolLua;
   MsSna_Workstation ref PathToWorkstation;
};
```

Properties

MsSna_PoolLua

Data Type: **ref PathToPoolLua** Qualifiers: **Key** Access Type: Read-Only

The LUA pool to associate with.

MsSna_Workstation

Data Type: **ref PathToWorkstation** Qualifiers: **Key** Access Type: Read-Only

The workstation to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ConnectionUsingAdapter Class

Associates a connection with an adapter.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ConnectionUsingAdapter : MsSna_Association
{
   MsSna_Adapter ref PathToAdapter;
   MsSna_Connection ref PathToConnection;
};
```

Properties

MsSna_Adapter

Data Type: **ref PathToAdapter** Qualifiers: **Key** Access Type: Read-Only

The adapter to associate with.

MsSna_Connection

Data Type: **ref PathToConnection** Qualifiers: **Key** Access Type: Read-Only

The connection to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_Lu3270AssignedToPool Class

Associates a 3270 LU with a pool.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_Lu3270AssignedToPool : MsSna_Association
{
   MsSna_Pool ref PathToPool3270;
   MsSna_Lu3270 ref PathToLu3270;
};
```

Properties

MsSna_Pool

Data Type: **ref PathToPool3270** Qualifiers: **Key** Access Type: Read-Only

The pool to associate with.

MsSna_Lu3270

Data Type: **ref PathToLu3270** Qualifiers: **Key** Access Type: Read-Only

The 3270 LU to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_PoolOnServer Class

Associates a pool with a server.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_PoolOnServer : MsSna_Association
{
   MsSna_Pool ref PathToPool3270;
   MsSna_Server ref PathToServer;
};
```

Properties

MsSna_Pool

Data Type: ref PathToPool3270 Qualifiers: **Key** Access Type: Read-Only

The pool to associate with.

MsSna_Server

Data Type: **ref PathToServer** Qualifiers: **Key** Access Type: Read-Only

The server to associate with.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Other Resources

WMISNA WMI Provider Classes

Administration and Management Programmer's Guide

MsSna_ExtendedStatus Class

Describes the extended status of a specified message.

The following syntax is simplified from MOF code.

Syntax

```
class MsSna_ExtendedStatus: __ExtendedStatus
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

MQBridge WMI Programmer's Reference

The MQBridge WMI Programmer's Reference describes the WMI provider classes that you can use to monitor the MSMQ-MQSeries Bridge.

For general information about programming for messaging, see theMessaging Programmer's Guide section of the SDK.

For sample programs illustratating MSMQ-MQSeries Bridge, see the Messaging Samples section of the SDK.

This section contains:

- WMIMQBridge WMI Provider Classes

WMIMQBridge WMI Provider Classes

The Microsoft® Host Integration Server MSMQ-MQSeries Bridge provider supplies information regarding the MSMQ-MQSeries Bridge. As an instance and method provider, the WMIMQBridge provider implements the standard **IWbemProviderInit** interface and the following **IWbemServices** methods:

- **CreateInstanceEnumAsync**
- **DeleteInstanceAsync**
- **ExecMethodAsync**
- **GetObjectAsync**
- **PutInstanceAsync**

For more information on **IWbemProviderInit** and **IWbemServices**, see "COM API for WMI" in the MSDN Library at http://msdn.microsoft.com/library.

You can access these provider classes in the \root\MicrosoftHIS namespace.

Class	Description
MsHisBridge_ExtendedStatus	Returns error information.
MsHisBridge_Service	Describes the MQSeries Bridge service.
MsHisBridge_Channel	Describes an MQI channel.
MsHisBridge_Channel_In_Service	Associates the channel and the service.
MsHisBridge_MessagePipe	Describes an MSMQ-MQSeries message pipe.
MsHisBridge_ConnectedNetwork	Describes a connected network.
MsHisBridge_ConnectedNetwork_In_Service	Associates the connected network to a specified service.

Requirements

Platforms: Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

MsHisBridge_ExtendedStatus Class

Returns error information.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisBridge_ExtendedStatus : __ExtendedStatus
{
};
```

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsHisBridge_Service Class

Describes the MQSeries Bridge service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisMQBridge_Service : MsHisMQBridge_Config
{
   string Name;
   uint32 F2QR_MaxThreads;
   uint32 F2QT_MaxThreads;
   uint32 Q2FD_MaxThreads;
   uint32 Q2FO_MaxThreads;
   uint32 CacheReresh;
   Boolean Encryption;
};
```

Properties

Name

Data Type: **String**

Qualifiers: **key**Access Type: Read/Write

The name of the computer.

F2QR_MaxThreads

Data Type: **uint32**

Access Type: Read/Write

The maximum number of threads for the MSMQ-to-MQSeries High service message pipe.

F2QT_MaxThreads

Data Type: **uint32**

Access Type: Read/Write

The maximum number of threads for the MSMQ-to-MQSeries Normal service message pipe.

Q2FD_MaxThreads

Data Type: **uint32**

Access Type: Read/Write

The maximum number of threads for the MQSeries-to-MSMQ High service message pipe.

Q2FO_MaxThreads

Data Type: **uint32**

Access Type: Read/Write

The maximum number of threads for the MQSeries-to-MSMQ Normal service message pipe.

CacheReresh

Data Type: **uint32**

Access Type: Read/Write

The interval (in minutes) at which the message pipes check for Cache Time-out.

Encryption

Data Type: **Boolean**

Access Type: Read/Write

Enables the Encryption feature from MSMQ to MSMQ-MQSeries Bridge.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsHisBridge_Channel Class

Describes an MQI channel.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisMQBridge_Channel : MsHisMQBridge_Config
{
   string Name;
   string QueueManagerName;
   uint32 TransportType;
   string TCPAddress;
   string TCPPort;
   string LU62SideInfoRecord;
   string McaUser;
};
```

Properties

Name

Data Type: **String**

Qualifiers: **key**Access Type: Read/Write

The name of the computer.

QueueManagerName

Data Type: **String**Access Type: Read/Write

The MQSeries Queue Manager to which the channel connects.

TransportType

Data Type: **uint32**Access Type: Read-Only

Specifies whether TCP/IP or SNA LU 6.2 communication will be used. The following table describes the possible values for **TransportType**.

Value	Description
1	LU 6.2
2	TCP/IP

TCPAddress

Data Type: **String**Access Type: Read/Write

The TCP/IP Address of the MQSeries listener (used with TCP/IP transport).

TCPPort

Data Type: **String**Access Type: Read/Write

The TCP/IP Port of the MQSeries listener (used with TCP/IP transport).

LU62SideInfoRecord

Data Type: **String**Access Type: Read/Write

The CPI-C Symbolic Name defined in Host Integration Server 2009 (used with SNA LU 6.2 transport).

McaUser

Data Type: **String**Access Type: Read/Write

An existing or new MQSeries user name, such as *FMQUSER1*, under which the server side of the MQI channel runs.

Requirements

Platforms: Microsoft Windows Server 2003, Windows XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsHisBridge_Channel_In_Service Class

Associates the channel and the service.

Syntax

```
class MsHisMQBridge_Channel_In_Service : MsHisMQBridge_Association
{
   MsHisMQBridge_Channel ref PathToChannel;
   MsHisMQBridge_Service ref PathToService;
};
```

Properties

PathToChannel

Data Type: **MsHisMQBridge_Channel ref**Qualifiers: **key**Access Type: Read-Only

The path to the channel.

PathToService

Data Type: **MsHisMQBridge_Service ref**Qualifiers: **key**Access Type: Read-Only

The path to the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

MsHisBridge_MessagePipe Class

Describes an MSMQ-MQSeries message pipe.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisMQBridge_MessagePipe
{
   string TransmitQueue;
   uint32 Startup;
   uint32 MaxMessageCount;
   uint32 MaxMessageSize;
   uint32 MaxTime;
   uint32 CacheReresh;
   uint32 ShortRetryCount;
   uint32 ShortRetryDelay;
   uint32 LongRetryCount;
   uint32 LongRetryDelay;
};
```

Properties

TransmitQueue

Data Type: **String**Access Type: Read/Write

A unique MQSeries transmission queue name for the pipe. The default names are *<CN name>.XMITQ* for the transactional message pipe, and *<CN name>.XMITQ.HIGH* for the nontransactional message pipe.

Startup

Data Type: **uint32**Access Type: Read/Write

Enabled or disabled at MSMQ-MQSeries Bridge Startup.

MaxMessageCount

Data Type: **uint32**Access Type: Read/Write

The maximum number of messages in a batch.

MaxMessageSize

Data Type: **uint32**Access Type: Read/Write

The maximum size (in bytes) of a batch.

MaxTime

Data Type: **uint32**Access Type: Read/Write

The interval (in minutes) at which the message pipes check for Cache Time-out.

CacheReresh

Data Type: **uint32**Access Type: Read/Write

The maximum time (in milliseconds) during which messages are batched.

ShortRetryCount

Data Type: **uint32**Access Type: Read/Write

The maximum number of retries for the short retry cycle.

ShortRetryDelay

Data Type: **uint32**Access Type: Read/Write

The interval between retries for the short retry cycle.

LongRetryCount

Data Type: **uint32**Access Type: Read/Write

The maximum number of retries for the long retry cycle.

LongRetryDelay

Data Type: **uint32**Access Type: Read/Write

The interval between retries for the long retry cycle.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsHisBridge_ConnectedNetwork Class

Describes a connected network.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisMQBridge_ConnectedNetwork : MsHisMQBridge_Config
{
   string Name;
   string MQSeriesQMNGR;
   string ReplyToQMNGR;
   uint32 Startup;
   MsHisMQBridge_MessagePipe MSMQ2MQSeries_MessagePipe;
   MsHisMQBridge_MessagePipe MSMQ2MQSeriesTx_MessagePipe;
   MsHisMQBridge_MessagePipe MQSeries2MSMQ_MessagePipe;
   MsHisMQBridge_MessagePipe MQSeries2MSMQTx_MessagePipe;
};
```

Properties

Name

Data Type: **String**

Qualifiers: **key**Access Type: Read/Write

The name of the computer.

MQSeriesQMNGR

Data Type: **String**Access Type: Read/Write

The MQSeries Queue Manager to which this CN is connected.

ReplyToQMNGR

Data Type: **String**Access Type: Read/Write

The default MSMQ QM to which MQSeries should return report or acknowledgment messages. Ordinarily, **ReplyToQMNGR** contains the name of the MSMQ-MQSeries Bridge computer.

Startup

Data Type: **uint32**Access Type: Read/Write

Enabled or disabled at MSMQ-MQSeries Bridge Startup.

MSMQ2MQSeries_MessagePipe

Data Type: **MsHisMQBridge_MessagePipe**Access Type: Read/Write

The MSMQ to MQ Series High service (nontransacted) message pipe.

MSMQ2MQSeriesTx_MessagePipe

Data Type: **MsHisMQBridge_MessagePipe**Access Type: Read/Write

The MSMQ to MQ Series Normal service (transacted) message pipe.

MQSeries2MSMQ_MessagePipe

Data Type: **MsHisMQBridge_MessagePipe**Access Type: Read/Write

The MQ Series to MSMQ High service (nontransacted) message pipe.

MQSeries2MSMQTx_MessagePipe

Data Type: **MsHisMQBridge_MessagePipe**Access Type: Read/Write

The MQ Series to MSMQ Normal service (transacted) message pipe.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

MsHisBridge_ConnectedNetwork_In_Service Class

Associates the connected network to a specified service.

The following syntax is simplified from MOF code.

Syntax

```
class MsHisMQBridge_ConnectedNetwork_In_Service : MsHisMQBridge_Association
{
   MsHisMQBridge_ConnectedNetwork ref PathToConnectedNetwork;
   MsHisMQBridge_Service ref PathToService;
};
```

Properties

PathToConnectedNetwork

Data Type: **MsHisMQBridge_ConnectedNetwork ref**

Qualifiers: **key**Access Type: Read-Only

The path to the connected network.

PathToService

Data Type: **MsHisMQBridge_Service ref**

Qualifiers: **key**Access Type: Read-Only

The path to the service.

Requirements

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

See Also

Reference

WMIMQBridge WMI Provider Classes

Other Resources

Administration and Management Programmer's Guide

Messaging Programmer's Reference

This section of the Microsoft Host Integration Server 2009 Developer's Guide lists the extensions and components that make up the MSMQ-MQSeries Bridge.

For general information about programming for the MSMQ-MQSeries Bridge, see the Messaging Programmer's Guide section of the SDK.

For sample code using the MSMQ-MQSeries Bridge, see the Messaging Programmer's Guide section of the SDK.

In This Section

- MSMQ-MQSeries Bridge Extensions Reference
- SDComponents for MSMQ-MQSeries Bridge Extensions

MSMQ-MQSeries Bridge Extensions Reference

This section provides an alphabetic reference to all of the API calls for the Microsoft® MSMQ-MQSeries Bridge Extension Property API.

In This Section

EPAdd

The **EPAdd** function adds a new extension field at the end of an existing **EP** object and optionally returns a cursor pointing to the new extension field in the **EP** object.

Syntax

```
HRESULT EPAdd(
        HANDLE hExtension,
    PCGUID pFieldID
        void *pFieldData,
        DWORD dwDataLength,
        PHANDLE phCursor
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle to the **EP** object that is to have data added.

pFieldID

Supplied parameter. A pointer to the GUID of the new extension field. A GUID is 16 bytes in length.

pFieldData

Supplied parameter. A pointer to the buffer containing the data for the new extension field.

dwDataLength

Supplied parameter. The length of the buffer containing the data for the new extension field.

phCursor

Supplied and returned parameter. A pointer to a cursor, which points to the new extension field. If *phCursor* is NULL when this function is called; the cursor is not created.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_ALLOC_FAIL

The function failed because memory could not be allocated for the internal data buffers used to extend the **EP** object.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

Remarks

In an **EP** object possessing a cursor, the extension fields are sorted in ascending order based on the GUID of each extension field. The message extension API functions may run more slowly while the cursor is in effect.

All cursors are canceled if the **EPDeleteAll** function is called with a *pFieldID* of NULL.

The following example illustrates how to use this function.

```
HANDLE hExt;
HANDLE hCursor;
GUID guid;
...
/* Add a new field containing the data "test" */
EPAdd(hExt, &guid, "test", 5, NULL);
```

```
/* Add a new field and create a cursor */
EPAdd(hExt, &guid, "another test", 13, &hCursor);
```

See Also

Reference

EPDelete

EPDeleteAll

EPClose

The **EPClose** function closes an open **EP** object freeing the extension handle and associated memory of an **EP** object. The entire contents of the object are deleted.

Syntax

```
HRESULT EPClose(
    PHANDLE phExtension
    );
```

Parameters

phExtension

Supplied and returned parameter. A pointer to an existing **EP** object handle to close.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because the parameter passed to this function is invalid.

Remarks

If the **EP** object handle is successfully closed, *phExtension* is reset to NULL on output.

See Also

Reference

EPOpen

EPDelete

The **EPDelete** function deletes a single extension field from an existing **EP** object.

Syntax

```
HRESULT EPDelete(
    HANDLE hExtension,
        PHANDLE phCursor
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle to the **EP** object that is to have extension field deleted.

phCursor

Supplied and returned parameter. On input, *phCursor* is a cursor pointing to the extension field to be deleted. On output, *phCursor* is set to the next extension field in the extension, or to NULL if there are no more fields.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

Remarks

After successful deletion, the cursor is set to point to the next field after the deleted one. If the last field is deleted, the cursor is set to NULL (to the beginning).

In an **EP** object possessing a cursor, the extension fields are sorted in ascending order based on the GUID of each extension field. The message extension API functions may run more slowly while the cursor is in effect.

All cursors are canceled if the **EPDeleteAll** function is called with a *pFieldID* of NULL.

See Also

Reference

EPDeleteAll

EPDeleteAll

The **EPDeleteAll** function deletes all extension fields from an existing **EP** object or all extension fields matching a specific GUID.

Syntax

```
HRESULT EPDeleteAll(
    HANDLEhExtension,
        PCGUIDpFieldsId,
        PHANDLEphCursor
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle to the **EP** object that is to have extension field deleted.

pFieldsId

Supplied parameter. A pointer to a 16-byte buffer containing the GUID of the fields to delete. If this parameter is NULL, all extension fields are deleted.

phCursor

Supplied and returned parameter. On output, this field is a pointer to an extension field cursor. The cursor is positioned at the first field of the next higher GUID after the one that was deleted. If there are no more GUIDs, or if all fields were deleted, *phCursor* is set to NULL. The *phCursor* may be NULL on input if no cursor is desired.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_EXTENSION_FIELD_NOT_FOUND

The function failed because the extension field matching the specified GUID could not be found.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

Remarks

If the *phCursor* parameter is not NULL, **phCursor* will point to the next field after all the deleted ones (even if MQ_ERROR_EXTENSION_FIELD_NOT_FOUND is returned). If the last field is deleted, the cursor is set to NULL.

In an **EP** object possessing a cursor, the extension fields are sorted in ascending order based on the GUID of each extension field. The message extension API functions may run more slowly while the cursor is in effect.

All cursors are canceled if the **EPDeleteAll** function is called with a *pFieldID* of NULL.

The following example illustrates how to use this function.

```
HANDLE hExt;
GUID guid;
HANDLE hCursor;
...
/* Delete all fields having a specified GUID and set a cursor */
EPDeleteAll(hExt, &guid, &hCursor);

/* Delete all extension fields */
EPDeleteAll(hExt, NULL, NULL);
```

See Also

Reference

EPDelete

EPGet

The **EPGet** function reads a specified extension field from an **EP** object, storing the GUID, length, and data subfields in separate variables. **EPGet** can also be used to locate extension fields containing a specified GUID

Syntax

```
HRESULT EPGet(
        HANDLE hExtension,
        NAVTYPE Directive,
        PHANDLE phCursor,
        GUID *pFieldID,
        void *pFieldData,
        PDWORD pdwDataLength,
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle.

Directive

Supplied parameter. This parameter and *phCursor* together control the behavior of the function. Possible values and their usage are discussed in the table following this parameter list.

phCursor

Supplied and returned parameter. A pointer to a cursor, which points to the matching extension field. If *phCursor* is NULL, the first field having a GUID matching *pFieldID* is read, and the cursor is positioned to this field. See the *Directive* argument in this list for specific details.

pFieldID

Supplied and returned parameter. If the *Directive* argument is EP_NEXT_KEY_FIELD, *pFieldId* is a pointer to a 16-byte buffer containing the GUID to be read. If the *Directive* argument is EP_CURRENT_FIELD or EP_NEXT_FIELD, on output this parameter is a pointer to a 16-byte buffer where the function stores the GUID. The GUID can be NULL if the GUID output is not desired.

pFieldData

Supplied and returned parameter. On input, a pointer to a buffer where the function should store the extension field data. On input, this parameter can be NULL if the data output is not desired. On output, a pointer to a buffer where the function stores the extension field data.

pdwDataLength

Supplied and returned parameter. On input, the length of the buffer for the data from the extension field. On output, the actual length of the data. If the buffer is too short or NULL, the data is not read but *pdwDataLength* is reset to the required buffer length.

Values for the Directive parameter

Directive	Description
EP_CURRENT_FIELD	Retrieves the extension field pointed to by the *phCursor* parameter, which must be a valid non-NULL cursor handle.

EP_NEXT_FIELD	Advances the cursor and reads the next field. If there are no more fields, returns MQ_ERROR_EXTENSION_FIELD_NOT_FOUND and *phCursor* is set to NULL. If *phCursor* is NULL or **phCursor* is NULL, the cursor is positioned to the first field (sorted in ascending order of GUID) and this field is read. If *phCursor* is not NULL, **phCursor* is set to the extension field. If *phCursor* is not NULL and **phCursor* is not NULL, the cursor (*phCursor*) is positioned to the first field (sorted in ascending order of GUID) and this field's ID and data are read and returned.
EP_NEXT_KEY_FIELD	Advances the cursor and reads the next field having a GUID matching *pFieldID*. If there are no more matching fields, returns MQ_ERROR_EXTENSION_FIELD_NOT_FOUND and sets phCursor to the first field having the next higher GUID, or to NULL if there are no more fields. If *phCursor* is NULL or **phCursor* is NULL, then the first field having a GUID matching *pFieldID* is read, and the cursor is positioned to this field. If the field is found, its field ID and data are returned and if *phCursor* is not NULL, **phCursor* is set to the field.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_ALLOC_FAIL

The function failed because memory could not be allocated for the internal data buffers used for the **EP** object.

MQ_ERROR_EXTENSION_FIELD_NOT_FOUND

The function failed because the extension field matching the specified GUID could not be found.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

MQ_ERROR_USER_BUFFER_TOO_SMALL

The function failed because the length of the buffer passed was too small for the data.

Remarks

In an **EP** object possessing a cursor, the extension fields are sorted in ascending order based on the GUID of each extension field. The message extension API functions may run more slowly while the cursor is in effect.

All cursors are canceled if the **EPDeleteAll** function is called with a *pFieldID* of NULL.

The following example illustrates how to use this function.

```
HANDLE hExt;
HANDLE hCursor;
GUID guid;
void *pBuffer;
DWORD dwSize, dwCount;
DWORD dwTotalSize = 0;
...

/* Retrieve one field by GUID */
dwSize = 1024;
EPGet(hExt, EP_NEXT_KEY_FIELD, NULL, &guid, pBuffer, &dwSize);

/* Count the fields in a message extension */
for (hCursor = NULL, dwCount = 0;
     EPGet(hExt, EP_NEXT_FIELD, &hCursor, NULL, NULL, NULL)==MQ_OK;
     dwCount++);

/* Compute the total length of all extension fields
   having a given GUID */
```

```
for (hCursor = NULL, dwCount = 0;
          EPGet(hExt, EP_NEXT_KEY_FIELD, &hCursor, &guid, NULL,
              &dwSize) == MQ_OK;
      dwTotalSize += dwSize);
```

See Also

Reference

EPDelete

EPDeleteAll

EPGetBuffer

The **EPGetBuffer** function converts an **EP** object to a message extension (PROPID_M_EXTENSION format) that can be sent in a message and packs the message extension into the supplied buffer.

Syntax

```
HRESULT EPGetBuffer(
    HANDLE hExtension,
        void *pBuf,
        PDWORD pdwBufLength,
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle.

pBuf

Supplied parameter. A pointer to the buffer where this function will store the PROPID_M_EXTENSION message extension.

pdwBufLength

Supplied and returned parameter. On input, the length of the buffer for the message extension. On output, the actual length of the stored data. If the buffer is too short or NULL, the data is not converted but *pdwBufLength* is reset to the required buffer length.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

MQ_ERROR_USER_BUFFER_TOO_SMALL

The function failed because the length of the buffer passed was too small for the data.

Remarks

If this function executed successfully, the *pdwBufLength* parameter contains the actual length of the packed extension buffer. If MQ_ERROR_USER_BUFFER_TOO_SMALL is returned, *pdwBufLength* contains the required buffer length.

The following example illustrates how to use this function.

```
HANDLE hExt;
void *pBuffer;
DWORD dwSize;

/* Read the required buffer length */
dwSize = 0;
EPGetBuffer(hExt, NULL, &dwSize);

/* Allocate the buffer */
pBuffer = malloc(dwSize);

/* Write the message extension to the buffer */
EPGetBuffer(hExt, pBuffer, &dwSize);
```

See Also

Reference

EPAdd
EPOpen

EPOpen

The **EPOpen** function creates an **EP** object and optionally unpacks the supplied message extension buffer into it.

Syntax

```
HRESULT EPOpen(
    PHANDLE phExtension,
        Void *pExtBuffer,
        DWORD dwExtBufLength
        );
```

Parameters

phExtension

Returned parameter. A pointer to an **EP** object handle of the **EP** object that is created.

pExtBuffer

Supplied parameter. The pointer to a buffer containing the message extension data in the sequence GUID (16 bytes), length of data (4 bytes), data, GUID, length of data, data, etc.

dwExtBufLength

Supplied parameter. The length of the buffer containing the message extension data.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_ALLOC_FAIL

The function failed because memory could not be allocated for the **EP** object handle and internal data buffers.

MQ_ERROR_CORRUPTED_EXTENSION_BUFFER

The function failed because the buffer containing the message extension data was corrupted.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

Remarks

If a NULL pointer is passed for *pExtBuffer* or *dwExtBufLength* is zero, an **EP** object with no extension fields is created.

The following example illustrates how to use this function.

```
HANDLE hExt1, hExt2;
void *pBuffer;
DWORD dwBufLength;
...
/* Create an empty EP object */
EPOpen(&hExt1, NULL, 0);
/* Create an EP object, copying data from a message extension */
EPOpen(&hExt2, pBuffer, dwBufLength);
```

See Also

Reference

EPClose

EPUpdate

The **EPUpdate** function updates (replaces) the data and length subfields of an existing extension field in an **EP** object.

Syntax

```
HRESULT EPUpdate(
    HANDLE hExtension,
        HANDLE hCursor
        void *pFieldData,
        DWORD dwDataLength,
    );
```

Parameters

hExtension

Supplied parameter. The **EP** object handle to the **EP** object that is to have data updated.

hCursor

Supplied parameter. The cursor pointing to the extension field to be updated which must not be NULL.

pFieldData

Supplied parameter. A pointer to the buffer containing the new data for the extension field. This parameter may be NULL for an empty data subfield.

dwDataLength

Supplied parameter. The length of the buffer containing the new data for the extension field.

Return Codes

MQ_OK

The function executed successfully.

MQ_ERROR_ALLOC_FAIL

The function failed because memory could not be allocated for the internal data buffers used to update the **EP** object.

MQ_ERROR_INVALID_HANDLE

The function failed because the **EP** object handle passed to the function is invalid.

MQ_ERROR_INVALID_PARAMETER

The function failed because one or more of the parameters passed to this function are invalid.

Remarks

The following example illustrates how to use this function.

```
HANDLE hExt;
HANDLE hCursor;
GUID guid;
...
/* Find an extension field containing a specified GUID */
EPGet (hExt, EP_NEXT_KEY_FIELD, &hCursor, &guid, NULL, 0)
/* Change the length and data subfields of the extension field */
EPUpdate(hExt, hCursor, "newdata", 8);
```

See Also

Reference

EPAdd

EPGet

SDComponents for MSMQ-MQSeries Bridge Extensions

The Host Integration Server 2009 SDK contains software components used for application integration using messaging and the MSMQ-MQSeries Bridge.

In This Section

Program and DLL Files for MSMQ-MQSeries Bridge

The following table lists the executable system files, DLL library files, and other files that are included with the Host Integration Server 2009 SDK for use with the MSMQ-MQSeries Bridge.

File name	Description
BCluster.exe	A program used to create or remove the MSMQ-MQSeries Bridge Service resource in a cluster.
explres.dll	The resource file for Q2QEXPL.exe, the MSMQ-MQSeries Bridge explorer program.
MQBInst.dll	The COM component used to create, delete, or modify MSMQ-MQSeries Bridge objects in Active Directory® directory service.
mqfrgnky.dll	A library to provide a function for the MSMQ-MQSeries Bridge to store a public key in a foreign computer object (for example, MQFrgn_StorePubKeysInDS).
MQSRRecv.exe	A sample program that uses the MQSeries API to receive messages from a specified MQSeries queue. This program can be used to test the operation of the MSMQ-MQSeries Bridge.
MQSRSend.exe	A sample program that uses the MQSeries API to send ten test messages to a specified MQSeries queue. This program can be used to test the operation of the MSMQ-MQSeries Bridge.
MSMQRecv.exe	A sample program that uses the Message Queuing (also known as MSMQ) API to receive messages from a specified Message Queuing queue. This program can be used to test the operation of the MSMQ-MQSeries Bridge.
MSMQSend.exe	A sample program that uses the Message Queuing API to sends 10 test messages to a specified Message Queuing local or foreign queue. This program can be used to test the operation of the MSMQ-MQSeries Bridge.
Q2QCLDLL.dll	A helper DLL (now obsolete) that contains functions to work with a cluster.
Q2QEXPL.exe	The MSMQ-MQSeries Bridge explorer program.
Q2QGW.exe	The MSMQ-MQSeries Bridge service program.
q2qmsg.dll	The Event Log message file.
q2qperf.ini	The performance counter definition file for the MSMQ-MQSeries Bridge.
q2qprfdl.dll	The performance counter implementation DLL for the MSMQ-MQSeries Bridge extension.
q2qprfsm.def	Defines the performance counter object in q2qperf.ini.
Q2QSHDLL.dll	A helper DLL (now obsolete) that contains functions for installing and uninstalling the MSMQ-MQSeries Bridge.
SHDLLRes.dll	The resource DLL (now obsolete) for Q2QSHDLL.DLL for installing and uninstalling the MSMQ-MQSeries Bridge.
wmiMQBridge.dll	The MSMQ-MQSeries Bridge Windows Management Instrumentation (WMI) Provider.

wmimqbridge.mof	The WMI Managed Object File (MOF) for the MSMQ-MQSeries Bridge.

Symbol Files for MSMQ-MQSeries Bridge

The following symbol files for use when debugging are included with Host Integration Server 2009 for use with the MSMQ-MQSeries Bridge. These files are installed as part of the Host Integration Server package and copies of these files are also located on the Host Integration Server CD under the Support\Symbols folder.

File name	Description
EXE\BCluster.dbg	Symbols from BCluster.exe
DLL\explres.dbg	Symbols from Explres.dll
DLL\MQBInst.dbg	Symbols from MQBInst.dll
EXE\MQSRRecv.dbg	Symbols from MQSRRecv.exe
EXE\MQSRSend.dbg	Symbols from MQSRSend.exe
EXE\MSMQRecv.dbg	Symbols from MSMQRecv.exe
EXE\MSMQSend.dbg	Symbols from MSMQSend.exe
DLL\Q2QCLDLL.dbg	Symbols from Q2QCLDLL.dll.
EXE\Q2QClIns.dbg	Symbols from Q2QClIns.exe.
EXE\Q2QClUni.dbg	Symbols from Q2QClUni.exe.
EXE\Q2QEXPL.dbg	Symbols from Q2QEXPL.exe.
EXE\Q2QGW.dbg	Symbols from Q2QGW.exe.
EXE\Q2QInst.dbg	Symbols from Q2QInst.exe.
DLL\q2qmsg.dbg	Symbols from q2qmsg.dll.
DLL\q2qprfdl.dbg	Symbols from q2qprfdl.dll.
DLL\Q2QSHDLL.dbg	Symbols from Q2QSHDLL.dll.
DLL\SHDLLRes.dbg	Symbols from SHDLLRes.dll.
DLL\wmiMQBridge.dbg	Symbols from wmiMQBridge.dll.

Header Files for MSMQ-MQSeries Bridge

Provider-specific header files needed to build the MSMQ-MQSeries Bridge sample applications are included with Host Integration Server 2009. These files are installed as part of the Host Integration Server package and copies of these files are also located on the Host Integration Server CD under the SDK\Include folder.

The following provider-specific files are provided with Host Integration Server for developing applications using the MSMQ-MQSeries Bridge.

File name	Description
msmqep.h	Globally unique identifier (GUID) definitions, enumeration constants, and error codes for use with the MSMQ-MQSeries Bridge.

Import Library Files for MSMQ-MQSeries Bridge

Provider-specific import library files needed to build the MSMQ-MQSeries Bridge extension sample applications are included with Host Integration Server 2009. These files are installed as part of the Host Integration Server package and copies of these files are also located on the Host Integration Server CD under the SDK\Lib folder.

The following provider-specific files are provided with Host Integration Server for developing applications using the MSMQ-MQSeries Bridge:

File name	Description
msmqep.lib	Import library of functions for use with the MSMQ-MQSeries Bridge extension.

See Also

Security Programmer's Reference

This section provides information required to develop applications which use the Enterprise Single Sign-On features integrated into Host Integration Server 2009.

Enterprise Single Sign-On (SSO) provides a way to map a Windows user ID to non-Windows user credentials. This service can simplify business processes that use applications on diverse systems.

In This Section

Single Sign-on Programmer's Reference

Single Sign-on Programmer's Reference

Enterprise Single Sign-On (SSO) is a technology that enables you to map a Microsoft Windows user ID to a non-Windows user credential, thus enabling your users to transparently log on to non-Windows servers.

In This Section

COM Mapper Programmer's Reference

Enterprise Single Sign-On Flags

Related Sections

Creating a Single Sign-On Application

Microsoft.EnterpriseSingleSignOn.Interop

COM Mapper Programmer's Reference

Component Object Model (COM) objects are used for accessing the Enterprise Single Sign-On (SSO) technology for Microsoft Host Integration Server.

In This Section

- IPropertyBag Interface
- ISSOAdmin Interface (COM)
- ISSOAdmin2 Interface (COM)
- ISSOConfigDB Interface (COM)
- ISSOConfigOM Interface (COM)
- ISSOConfigSS Interface (COM)
- ISSOConfigStore Interface (COM)
- ISSOLookup1 Interface (COM)
- ISSOLookup2 Interface (COM)
- ISSOMapper Interface (COM)
- ISSOMapper2 Interface (COM)
- ISSOMapping Interface (COM)
- ISSONotification Interface (COM)
- ISSOTicket Interface (COM)
- SAdapter Structure (COM)
- SExternalAccount Structure (COM)
- SPasswordChange Structure (COM)
- SPasswordChangeComplete Structure (COM)
- SSO_NOTIFICATION_TYPE Enumeration (COM)
- SSO_NOTIFICATION_FLAG Enumeration (COM)
- SStatus Structure (COM)

See Also

Other Resources

Single Sign-on Programmer's Reference

IPropertyBag Interface

The IPropertyBag interface acts as a parameter for ISSOConfigStore, and also allows you to change the behavior of specific interfaces in the Enterprise Single Sign-On (SSO) object model.

Requirements

Type Library: SSOAdminLib 1.0 Type Library (SSOAdminLib.dll), SSOConfigStoreLib 1.0 Type Library (SSOConfigStoreLib), and SSOPSAdmin 1.0 Type Library (SSOPSAdmin.dll)

Platforms: Microsoft® Windows Server™ 2003, Windows® XP Professional, Windows 2000 Server

IPropertyBag Members

The member of the IPropertyBag interface is described in the following table.

Public Members

Member	Description
IPropertyBag.RemoteRead Method	Reads the specified Single Sign-On property.
IPropertyBag.Write Method	Writes the specified Single Sign-on property.

See Also

Concepts

IPropertyBag Interface

IPropertyBag Methods

The method of the IPropertyBag interface is described in the following table. For a complete list of IPropertyBag interface members, see IPropertyBag Members.

Public Methods

Method	Description
IPropertyBag.RemoteRead Method	Reads the specified Single Sign-On property.
IPropertyBag.Write Method	Writes the specified Single Sign-on property.

See Also

Concepts

IPropertyBag Interface

IPropertyBag.RemoteRead Method

Writes the specified Single Sign-on (SSO) property.

Syntax
C#

```
void RemoteRead(
string pszPropName,
[out] object pVar,
SSOAdminLib.IErrorLog pErrorLog,
uint varType,
object pUnkObj
);
```

Parameters

Parameters	Description
pszPropName	The property name to read.
pVar	When returned, the object containing the property value.
pErrorLog	An IErrorLog containing the error log, if applicable.
varType	The object type of the property.
pUnkObj	Pointer to the unknown object to read.

Remarks

You can use IPropertyBag.Read to read the current settings for the SSO object model.

See Also
Concepts
IPropertyBag Methods

IPropertyBag.Write Method

Writes the specified Single Sign-on property.

Syntax
C#

```
void Write(
      string pszPpropName,
      object pVar
)
```

Parameters

Parameters	Reference
pszPropName	The name of the property to write.
pVar	The value of the property to write.

Remarks

- If you call QueryInterface on an SSO object, you can retrieve the IPropertyBag interface and use it to change the behavior of your current object.

See Also
Reference
Enterprise Single Sign-On Flags
Concepts
IPropertyBag Methods

ISSOAdmin Interface (COM)

The **ISSOAdmin** interface provides administration functions for the Enterprise Single Sign-On (SSO) server database.

Requirements

Type Library: SSOAdmin 1.0 Type Library (SSOAdmin.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin Members

The following table shows the IISOAdmin members.

Public Methods

Member	Description
CreateApplication	Creates an application in the Single Sign-On (SSO) server database.
CreateFieldInfo	Creates field information for an application.
DeleteApplication	Deletes an application from the SSO server database.
GetApplicationInfo	Gets the application information from the SSO server database.
GetGlobalInfo	Returns the global SSO server configuration information from the SSO server database.
PurgeCacheForApplication	Purges the cached credentials for an application on all SSO servers.
UpdateApplication	Updates the application information in the SSO server database.

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Methods

The methods of the **ISSOAdmin** interface are listed in the following table. For a complete list of **ISSOAdmin** interface members, see ISSOAdmin Members.

Public Methods

Method	Description
CreateApplication	Creates an application in the Single Sign-On (SSO) server database.
CreateFieldInfo	Creates field information for an application.
DeleteApplication	Deletes an application from the SSO server database.
GetApplicationInfo	Gets the application information from the SSO server database.
GetGlobalInfo	Returns the global SSO server configuration information from the SSO server database.
PurgeCacheForApplication	Purges the cached credentials for an application on all SSO servers.
UpdateApplication	Updates the application information in the SSO server database.

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin.CreateApplication Method

The **CreateApplication** method creates an application in the Enterprise Single Sign-On (SSO) server database.

Syntax
C++

```
HRESULT CreateApplication(
BSTR bstrApplicationName,
BSTR bstrDescription,
BSTR bstrContactInfo,
BSTR bstrUserGroupName,
BSTR bstrAdminGroupName,
LONG lFlags,
LONG lNumFields
);
```

```
[Visual Basic]
Sub CreateApplication(
bstrApplicationName As String,
bstrDescription As String,
bstrContactInfo As String,
bstrUserGroupName As String,
bstrAdminGroupName As String,
lFlags As Long,
lNumFields As Long
)
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrDescription

[in] String that specifies the description for the application. This parameter can be NULL, an empty string, or contain spaces.

bstrDescription

[in] String that specifies the description for the application. This parameter can be NULL, an empty string, or contain spaces.

bstrContactInfo

[in] String that specifies the contact information for this application. This parameter can be NULL, an empty string, or contain spaces.

bstrContactInfo

[in] String that specifies the contact information for this application. This parameter can be NULL, an empty string, or contain spaces.

bstrUserGroupName

[in] String that specifies the application users group name. This parameter must contain a valid global group.

bstrUserGroupName

[in] String that specifies the application users group name. This parameter must contain a valid global group.

bstrAdminGroupName

[in] String that specifies the Application Administrator group name. This parameter must contain a valid global group.

bstrAdminGroupName

[in] String that specifies the Application Administrator group name. This parameter must contain a valid global group.

lFlags

[in] Long integer that specifies whether the application is a group application. If a group application is required, specify SSO_FLAG_APP_USES_GROUP_MAPPING.

lFlags

[in] Long integer that specifies whether the application is a group application. If a group application is required, specify SSO_FLAG_APP_USES_GROUP_MAPPING.

lNumFields

[in] Long integer that specifies the number of fields that will be added for this application. The minimum value is 1 (one external user ID). An application can have no credential fields.

lNumFields

[in] Long integer that specifies the number of fields that will be added for this application. The minimum value is 1 (one external user ID). An application can have no credential fields.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

An application is always created as disabled.

After the field information is added by using the **CreateFieldInfo** method, then the application can be enabled by using the **UpdateApplication** method. The number of fields added by **CreateFieldInfo** must match the number of fields specified by the *numFields* parameter. The *numFields* value cannot be changed after the application is created.

To access this method, you must be an SSO Administrator or an SSO Affiliate Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin.CreateFieldInfo Method

The **CreateFieldInfo** method creates field information for an application.

Syntax
C++

```
HRESULT CreateFieldInfo(
BSTR bstrApplicationName,
BSTR bstrLabel,
LONG lFlags
);
```

VB

```
Sub CreateFieldInfo(
bstrApplicationName As String,
bstrLabel As String,
lFlags As Long
)
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrLabel

[in] String that specifies the label value. This parameter cannot be NULL or an empty string.

bstrLabel

[in] String that specifies the label value. This parameter cannot be NULL or an empty string.

lFlags

[in] Long integer specifies whether the field is masked. If a field must be masked when displayed in the user interface, specify SSO_FLAG_FIELD_INFO_MASK. The flag parameter will be ignored for the first (user ID) field, as it will not be masked.

lFlags

[in] Long integer specifies whether the field is masked. If a field must be masked when displayed in the user interface, specify SSO_FLAG_FIELD_INFO_MASK. The flag parameter will be ignored for the first (user ID) field, as it will not be masked.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.

E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

The application must exist before its field information can be created. The number of fields added must equal the *numFields* value specified when the application was created, otherwise the application will be disabled at run time.

Although the external user ID is not considered to be an external credential, it requires a field to describe how it will be displayed by the user interface. The first field created will be considered as the field that describes the external user ID. A minimum of one field is required, and typically at least two fields should be specified to provide credentials. An application can have no credentials.

To access this method, you must be an SSO Administrator, SSO Affiliate Administrator, or an SSO Application Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin.DeleteApplication Method

The **DeleteApplication** method deletes an application from the Enterprise Single Sign-On (SSO) server database.

Syntax
C++

```
HRESULT DeleteApplication(
BSTR bstrApplicationName
);
```

VB

```
Sub DeleteApplication(
bstrApplicationName As String
)
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
S_FALSE	The application was not found.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

All fields, mappings, and external credentials associated with this application are deleted.

To access this method, you must be an SSO Administrator or an SSO Affiliate Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin.GetApplicationInfo Method

The **GetApplicationInfo** method gets the application information from the Enterprise Single Sign-On (SSO) server database.

Syntax

C++

```
HRESULT GetApplicationInfo(
BSTR bstrApplicationName,
BSTR* pbstrDescription,
BSTR* pbstrContactInfo,
BSTR* pbstrUserGroupName,
BSTR* pbstrAdminGroupName,
LONG* plFlags,
LONG* plNumFields
);
```

VB

```
Function GetApplicationInfo(
bstrApplicationName As String,
pbstrDescription As String,
pbstrContactInfo As String,
pbstrUserGroupName As String,
pbstrAdminGroupName As String,
plFlags As Long
)
As Long
```

Parameters

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

pbstrDescription

[out] Pointer to a string that receives the description for the application.

pbstrDescription

[out] String that receives the description for the application.

pbstrContactInfo

[out] Pointer to a string that receives the contact information for the application.

pbstrContactInfo

[out] String that receives the contact information for the application.

pbstrUserGroupName

[out] Pointer to a string that receives the application user group name.

pbstrUserGroupName

[out] String that receives the application user group name.

pbstrAdminGroupName

[out] Pointer to a string that receives the Application Administrator group name.

pbstrAdminGroupName

[out] String that receives the Application Administrator group name.

plFlags

[out] Pointer to a long integer that receives the flags currently set for the application.

plFlags

[out] Long that receives the flags currently set for the application.

plNumFields

[out] Pointer to a long integer that receives the number of fields associated with the application.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

This method returns a Long that receives the number of fields associated with the application.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an SSO Administrator, SSO Affiliate Administrator, or an SSO Application Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin.GetGlobalInfo Method

The **GetGlobalInfo** method gets the global Enterprise Single Sign-On (SSO) configuration information from the Enterprise Single Sign-On server database.

Syntax

C++

```
HRESULT GetGlobalInfo(
LONG* plFlags,
LONG* plAuditAppDeleteMax,
LONG* plAuditMappingDeleteMax,
LONG* plAuditNtpLookupMax,
LONG* plAuditXpLookupMax,
LONG* plTicketTimeout,
LONG* plCredCacheTimeout,
BSTR* pbstrSecretServer,
BSTR* pbstrSSOAdminGroup,
BSTR* pbstrAffiliateAppMgrGroup
);
```

VB

```
Function GetGlobalInfo(
plFlags As Long,
plAuditAppDeleteMax As Long,
plAuditMappingDeleteMax As Long,
plAuditNtpLookupMax As Long,
plAuditXpLookupMax As Long,
plTicketTimeout As Long,
plCredCacheTimeout As Long,
pbstrSecretServer As String,
pbstrSSOAdminGroup As String
)
As String
```

Parameters

plFlags

[out] Pointer to a long integer that receives the global flags.

plFlags

[out] Long that receives the global flags.

plAuditAppDeleteMax

[out] Pointer to a long integer that receives the current size of the AuditAppDelete table.

plAuditAppDeleteMax

[out] Long integer that receives the current size of the AuditAppDelete table.

plAuditMappingDeleteMax

[out] Pointer to a long integer that receives the current size of the AuditMappingDelete table.

plAuditMappingDeleteMax

[out] Long that receives the current size of the AuditMappingDelete table.

plAuditNtpLookupMax

[out] Pointer to a long integer that receives the current size of the AuditNtpLookup table.

plAuditNtpLookupMax

[out] Long that receives the current size of the AuditNtpLookup table.

plAuditXpLookupMax

[out] Pointer to a long integer that receives the current size of the AuditXpLookup table.

plAuditXpLookupMax

[out] Long that receives the current size of the AuditXpLookup table.

plTicketTimeout

[out] Pointer to a long integer that receives the current ticket time-out value in minutes.

plTicketTimeout

[out] Long that receives the current ticket time-out value in minutes.

plCredCacheTimeout

[out] Pointer to a long integer that receives the current credential cache time-out value in minutes.

plCredCacheTimeout

[out] Long that receives the current credential cache time-out value in minutes.

pbstrSecretServer

[out] Pointer to a string that receives the current secret server computer name.

pbstrSecretServer

[out] String that receives the current secret server computer name.

pbstrSSOAdminGroup

[out] Pointer to a string that receives the current SSO server Administrator group name.

pbstrSSOAdminGroup

[out] String that receives the current SSO server Administrator group name.

pbstrAffiliateAppMgrGroup

[out] Pointer to a string that receives the current SSO server Affiliate Administrator group name.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

This method returns a String that receives the current SSO server Affiliate Administrator group name.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an SSO Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)
ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin.PurgeCacheForApplication Method

The **PurgeCacheForApplication** method purges the cached credentials for an application on all Enterprise Single Sign-On (SSO) servers.

Syntax

C++

```
HRESULT PurgeCacheForApplication(
BSTR bstrApplicationName
);
```

VB

```
Sub PurgeCacheForApplication(
bstrApplicationName As String
)
```

Parameters

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

There may be a short time delay while this operation is applied to all SSO servers.

To access this method, you must be an SSO Administrator, SSO Affiliate Administrator, or an Application Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

ISSOAdmin.UpdateApplication Method

The **UpdateApplication** method updates the application information in the Enterprise Single Sign-On (SSO) server database.

Syntax

C++

```
HRESULT UpdateApplication(
BSTR bstrApplicationName,
BSTR bstrDescription,
BSTR bstrContactInfo,
BSTR bstrUserGroupName,
BSTR bstrAdminGroupName,
LONG lFlags,
LONG lFlagMask
);
```

VB

```
Function UpdateApplication(
bstrApplicationName As String,
bstrDescription As String,
bstrContactInfo As String,
bstrUserGroupName As String,
bstrAdminGroupName As String,
lFlags As Long
)
```

Parameters

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrDescription

[in] String that specifies the description of the application.

bstrDescription

[in] String that specifies the description of the application.

bstrContactInfo

[in] String that specifies the contact information for this application.

bstrContactInfo

[in] String that specifies the contact information for this application.

bstrUserGroupName

[in] String that specifies the Application Users group name. This must be a valid global group.

bstrUserGroupName

[in] String that specifies the Application Users group name. This must be a valid global group.

bstrAdminGroupName

[in] String that specifies the Application Administrator group name. This must be a valid global group. If running as an

Application Administrator, this parameter will be ignored.

bstrAdminGroupName

[in] String that specifies the Application Administrator group name. This must be a valid global group. If running as an Application Administrator, this parameter will be ignored.

lFlags

[in] Integer that specifies whether the application is enabled. To enable the application, set the SSO_FLAG_ENABLED flag.

lFlags

[in] Integer that specifies whether the application is enabled. To enable the application, set the SSO_FLAG_ENABLED flag.

lFlagMask

[in] To change the flag value, set this mask to the flag that you need to change. For example, to enable or disable Enterprise SSO, set this flag to SSO_FLAG_ENABLED. The *flagMask* parameter indicates which flag you want to change, while the *flags* parameter indicates the new value of that flag.

lFlagMask

[in] To change the flag value, set this mask to the flag that you need to change. For example, to enable or disable Enterprise SSO, set this flag to SSO_FLAG_ENABLED. The *flagMask* parameter indicates which flag you want to change, while the *flags* parameter indicates the new value of that flag.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global **Err** object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an SSO Administrator, SSO Affiliate Administrator, or an Application Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOAdmin Interface (COM)

ISSOAdmin Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOAdmin2 Interface (COM)

The **ISSOAdmin2** interface provides additional administration functions for the Enterprise Single Sign-On (SSO) server database.

Requirements

Type Library: SSOAdmin 1.0 Type Library (SSOAdmin.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

ISSOAdmin2 Members

The following table shows the **IISOAdmin2** members.

Public Methods

Member	Description
ISSOAdmin.CreateApplication Method	Creates an application in the SSO server database.
ISSOAdmin.CreateFieldInfo Method	Creates field information for an application.
ISSOAdmin.DeleteApplication Method	Deletes an application from the SSO server database.
ISSOAdmin.GetApplicationInfo Method	Gets the application information from the SSO server database.
ISSOAdmin.GetGlobalInfo Method	Returns the global SSO server configuration information from the SSO server database.
ISSOAdmin.PurgeCacheForApplication Method	Purges the cached credentials for an application on all SSO server servers.
ISSOAdmin.UpdateApplication Method	Updates the application information in the SSO server database.
ISSOAdmin2.GetApplicationInfo2 Method	Gets the application information from the Enterprise Single Sign-On (SSO) server database.
ISSOAdmin2.UpdateApplication2 Method	Updates the application information in the Enterprise Single Sign-On (SSO) server database.

ISSOAdmin2 Methods

The following table shows the **IISOAdmin2** methods.

Public Methods

Method	Description
ISSOAdmin2.GetApplicationInfo2 Method	Gets the application information from the Enterprise Single Sign-On (SSO) server database.
ISSOAdmin2.UpdateApplication2 Method	Updates the application information in the Enterprise Single Sign-On (SSO) server database.

ISSOAdmin2.GetApplicationInfo2 Method

The **GetApplicationInfo2** method gets the application information from the Enterprise Single Sign-On (SSO) server database.

Syntax
VB

```
GetApplicationInfo2(
applicationName As String,
appInfoProps As object
)
```

C++

```
HRESULT GetApplicationInfo2(
BSTR applicationName,
IPropertyBag* appInfoProps
)
```

Parameters

applicationName

String containing the name of the application.

appInfoProps

IPropertyBag containing additional application information properties. For more information, see below.

Property Value/ Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

Remarks

The following table describes the accepted values for appInfoProps:

propName	Type	ptrValue
Contact	VT_BSTR	Contact name
Computer	VT_BSTR	Computer name
appAdminAccount	VT_BSTR	Application admin account

appUserAccount	VT_BSTR	Application user account
windowsAccount	VT_BSTR	Windows account
appTicketTimeout	VT_UI4	Application ticket timeout

In addition, individual flags may also use the following values:

propName	**Type**	**ptrValue**
enableApp	VT_BOOL	Enable application
hostInitiatedSSO	VT_BOOL	Host initiated SSO
validatePassword	VT_BOOL	Validate password
allowTickets	VT_BOOL	Allow tickets
syncFromAdapter	VT_BOOL	Synchronize from adapter
syncToAdapter	VT_BOOL	Synchronize to adapter
changeWindowsPassword	VT_BOOL	Change windows password
verifyOldPassword	VT_BOOL	Verify old password
sendOldPassword	VT_BOOL	Send old password
allowMappingConflicts	VT_BOOL	Allow mapping conflicts
groupApp	VT_BOOL	Group application
groupAdapter	VT_BOOL	Group adapter
allowLocalAccounts	VT_BOOL	Allow local accounts
adminAccountSame	VT_BOOL	Administration account same
configStoreApp	VT_BOOL	Config store application
timeoutTickets	VT_BOOL	Timeout tickets
directPasswordSync	VT_BOOL	Direct password synchronization
windowsCreds	VT_BOOL	Windows credentials
restrictedCreds	VT_BOOL	Restricted credentials
showFilterOnly	VT_BOOL	Show filter only
restrictMappingCreate	VT_BOOL	Restrict mapping create
windowsInitiatedSSO	VT_BOOL	Windows-initiated SSO

disableCredCache	VT_BOOL	Disable credentials cache

ISSOAdmin2.UpdateApplication2 Method

The UpdateApplication2 method updates the application information in the Enterprise Single Sign-On (SSO) server database.

Syntax
VB

```
UpdateApplication2(
applicationName As String,
appInfoProps As object
);
```

C++

```
void UpdateApplication2(
string applicationName,
IPropertyBag appInfoProps
);
```

Parameters

applicationName

String containing the new application name.

appInfoProps

IPropertyBag containing additional application information properties. For more information, see below.

Property Value/ Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

Remarks

The following table describes the accepted values for appInfoProps:

propName	**Type**	**ptrValue**
Contact	VT_BSTR	Contact name
Computer	VT_BSTR	Computer name
appAdminAccount	VT_BSTR	Application admin account
appUserAccount	VT_BSTR	Application user account

windowsAccount	VT_BSTR	Windows account
appTicketTimeout	VT_UI4	Application ticket timeout

In addition, individual flags may also use the following values:

propName	**Type**	**ptrValue**
enableApp	VT_BOOL	Enable application
hostInitiatedSSO	VT_BOOL	Host initiated SSO
validatePassword	VT_BOOL	Validate password
allowTickets	VT_BOOL	Allow tickets
syncFromAdapter	VT_BOOL	Synchronize from adapter
syncToAdapter	VT_BOOL	Synchronize to adapter
changeWindowsPassword	VT_BOOL	Change windows password
verifyOldPassword	VT_BOOL	Verify old password
sendOldPassword	VT_BOOL	Send old password
allowMappingConflicts	VT_BOOL	Allow mapping conflicts
groupApp	VT_BOOL	Group application
groupAdapter	VT_BOOL	Group adapter
allowLocalAccounts	VT_BOOL	Allow local accounts
adminAccountSame	VT_BOOL	Administration account same
configStoreApp	VT_BOOL	Config store application
timeoutTickets	VT_BOOL	Timeout tickets
directPasswordSync	VT_BOOL	Direct password synchronization
windowsCreds	VT_BOOL	Windows credentials
restrictedCreds	VT_BOOL	Restricted credentials
showFilterOnly	VT_BOOL	Show filter only
restrictMappingCreate	VT_BOOL	Restrict mapping create
windowsInitiatedSSO	VT_BOOL	Windows-initiated SSO
disableCredCache	VT_BOOL	Disable credentials cache

In addition, you may directly specify flags using the following properties:

propName	Type	ptrValue
flags	VT_UI4	Flags to specify
flagsMask	VT_UI4	Flag mask to specify

ISSOConfigDB Interface (COM)

The **ISSOConfigDB** object allows you to create and configure an SSO database.

Type Library: SSOConfigDB 1.0 Type Library (SSOConfigDB.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

ISSOConfigDB Members

The following table describes the **ISSOConfig DB** members.

Method	Description
ISSOConfigDB.CreateDatabase Method	Creates a SQL database for SSO.
ISSOConfigDB.GetDBInfo Method	Retrieves information about the specified database.
ISSOConfigDB.UpgradeDB Method	Upgrades the specified Single Sign-On (SSO) database to SSO version 3.

ISSOConfigDB Methods

The following table describes the **ISSOConfig DB** methods

Method	Description
ISSOConfigDB.CreateDatabase Method	Creates a SQL database for SSO.
ISSOConfigDB.GetDBInfo Method	Retrieves information about the specified database.
ISSOConfigDB.UpgradeDB Method	Upgrades the specified Single Sign-On (SSO) database to SSO version 3.

ISSOConfigDB.CreateDatabase Method

The CreateDatabase method creates a SQL database for SSO.

Syntax
VB

```
Sub CreateDatabase(
sqlServer As String,
sqlDatabase As String,
configureOnly As Boolean,
secretServer As String,
ssoAdminAccount As String,
ssoAffilateAdminAccount As String
);
```

C++

```
HRESULT CreateDatabase(
BSTR sqlServer,
BSTR sqlDatabase,
bool configureOnly,
BSTR secretServer,
BSTR ssoAdminAccount,
BSTR ssoAffilateAdminAccount
);
```

Parameters

Parameter	Description
sqlServer	String containing the name of the SQL server for the database.
sqlDatabase	String containing the name of the SQL database.
configureOnly	True to configure an already-existing database; otherwise, false.
secretServer	String containing the name of the secret server.
ssoAdminAccount	String containing the name of the SSO administration account for the server.
ssoAffilateAdminAccount	String containing the name of the SSO affiliate administration account.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.

E_INVALIDREG	An invalid parameter was detected.

ISSOConfigDB.GetDBInfo Method

The GetDBInfo method retrieves information about the specified database.

Syntax
VB

```
Sub GetDBInfo(
sqlServer As String,
sqlDatabase As String,
exists As Boolean,
isConfigured As Boolean,
needsUpgrade As Boolean,
secretServer As String,
ssoAdminAccount As String,
ssoAffilateAdminAccount As String
);
```

C++

```
HRESULT GetDBInfo(
string sqlServer,
string sqlDatabase,
out bool exists,
out bool isConfigured,
out bool needsUpgrade,
out string secretServer,
out string ssoAdminAccount,
out string ssoAffilateAdminAccount
);
```

Parameters

Parameter	Description
sqlServer	String containing the name of the SQL server.
sqlDatabase	String containing the name of the specified database.
exists	Returns true if the database exists; otherwise, false.
isConfigured	Returns true if the database is configured; otherwise, false.
needsUpgrade	Returns true if the database needs to be upgraded; otherwise, false.
secretServer	Returns a string containing the name of the secret server of the database.
ssoAdminAccount	Returns a string containing the name of the SSO administration account.
ssoAffilateAdminAccount	Returns a string containing the SSO affiliate administration account.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigDB.UpgradeDB Method

The UpgradeDatabase method upgrades the specified Single Sign-On (SSO) database to SSO version 3.

Syntax
VB

```
Sub UpgradeDB (
sqlServer As String,
sqlDatabase As String
);
```

C++

```
HRESULT UpgradeDB (
BSTR sqlServer,
BSTR sqlDatabase
);
```

Parameters

Parameter	Description
sqlServer	String containing the SQL server in which the database exists.
sqlDatabase	String containing the name of the database.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigOM Interface (COM)

The **ISSOConfigOM** grants access to the Single Sign-On (SSO) object model for server configuration.

Requirements

Type Library: SSOConfigOM 1.0 Type Library (SSOConfigOM.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

ISSOConfigOM Members

The following table shows the **ISSOConfigOM** members.

Public Methods

Method	Description
ISSOConfigOM.DiscoverServer Method	Discovers the currently-available servers.
ISSOConfigOM.GetServerStatus	Describes the status of the current server.

ISSOConfigOM Methods

The following table shows the **ISSOConfigOM** methods.

Public Methods

Method	Description
ISSOConfigOM.DiscoverServer Method	Discovers the currently-available servers.
ISSOConfigOM.GetServerStatus	Describes the status of the current server.

ISSOConfigOM.DiscoverServer Method

The **DiscoverServers** method discovers the currently-available servers.

Syntax
VB

```
  Servers As String
  );
```

C#

```
  BSTR Servers
  );
```

Parameters

Parameter	Reference
Servers	Returns a string containing the currently-available servers.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigOM.GetServerStatus

The **GetServerStatus** method describes the status of the current server.

Syntax
VB

```
GetServerStatus(
flags As Integer,
ssoServerName As String,
sqlServer As String,
sqlDatabase As String,
serviceAccount As String,
computerNameCluster As String,
computerNameNode As String,
eventCountInformational As Integer,
eventCountWarning As Integer,
eventCountError As Integer,
versionInfoM As Integer,
versionInfoL As Integer,
auditLevelN As Integer,
auditLevelP As Integer,
passwordSyncAge As Integer,
statusFlags As Integer
);
```

C++

```
HRESULT GetServerStatus(
int flags,
BSTR ssoServerName,
BSTR sqlServer,
BSTR sqlDatabase,
BSTR serviceAccount,
BSTR computerNameCluster,
BSTR computerNameNode,
BSTR eventCountInformational,
int eventCountWarning,
int eventCountError,
int versionInfoM,
int versionInfoL,
int auditLevelN,
int auditLevelP,
int passwordSyncAge,
int statusFlags
);
```

Parameters

Parameter	Description
flags	Not used in this version. Should be set to zero.
ssoServerName	Returns a string containing the current SSO server name.
sqlServer	Returns a string containing the name of the SQL server of the current server.
sqlDatabase	Returns a string containing the name of the SQL database of the current server.
serviceAccount	Returns a string containing the current service account.

computerNameCluster	Returns a string containing the name of the current computer cluster.
computerNameNode	Returns the name of the current computer.
eventCountInformational	Returns an integer containing information regarding the event count.
eventCountWarning	Returns an integer containing the event count warning.
eventCountError	Returns an integer containing the event count error.
versionInfoM	Returns an integer containing the MSB version info.
versionInfoL	Returns an integer containing the LSB version info.
auditLevelN	Returns an integer containing the negative audit level.
auditLevelP	Returns an integer containing the positive audit level.
passwordSyncAge	Returns an integer containing the password sync age.
statusFlags	Returns an integer containing the status flags. For more information, see SSOStatusFlags.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigSS Interface (COM)

The **ISSOConfigSS** interface configures the secret server.

Requirements

Type Library: SSOConfigSS 1.0 Type Library (SSOConfigSS.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

ISSOConfigSS Members

The following table describes the **ISSOConfigSS** members.

Method	Description
ISSOConfigSS.BackupSecret Method	Backs up the secret server.
ISSOConfigSS.GenerateSecret Method	Generates the secret for the secret server.
ISSOConfigSS.GetFilePasswordReminder Method	Retrieves the file password reminder for the secret server.
ISSOConfigSS.RestoreSecret Method	Restores the secret from the specified restore file.

ISSOConfigSS Methods

The following table describes the **ISSOConfigSS** methods.

Method	Description
ISSOConfigSS.BackupSecret Method	Backs up the secret server.
ISSOConfigSS.GenerateSecret Method	Generates the secret for the secret server.
ISSOConfigSS.GetFilePasswordReminder Method	Retrieves the file password reminder for the secret server.
ISSOConfigSS.RestoreSecret Method	Restores the secret from the specified restore file.

ISSOConfigSS.BackupSecret Method

The **BackupSecret** method backs up the secret server.

Syntax
VB

```
  backupFile As String,
  filePassword As String,
  filePasswordReminder As String
  );
```

C++

```
  BSTR backupFile,
  BSTR filePassword,
  BSTR filePasswordReminder
  );
```

Parameters

Parameter	**Description**
backupFile	String containing the path and name of the secret server backup file.
filePassword	String containing the backup file password.
filePasswordReminder	String containing the secret server file password reminder.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigSS.GenerateSecret Method

The **GenerateSecret** method generates the secret for the secret server.

Syntax
VB

```
GenerateSecret(
backupFile As String,
filePassword As String,
filePasswordReminder As String
);
```

C++

```
HRESULT GenerateSecret(
BSTR backupFile,
BSTR filePassword,
BSTR filePasswordReminder
);
```

Parameters

Parameter	**Description**
backupFile	String containing the path and name of the backup file for the secret server.
filePassword	String containing the backup file password.
filePasswordReminder	String containing the secret server file password reminder

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOConfigSS.GetFilePasswordReminder Method

The **GetFilePasswordReminder** method gets the password reminder from the backup file.

Syntax
VB

```
Function GetFilePasswordReminder(
restoreFile as String);
```

C#

```
void GetFilePasswordReminder(
string restoreFile,
out string filePasswordReminder);
```

Parameters

Parameter	Description
restoreFile	The file from which the reminder is to be retrieved.
filePasswordReminder	The reminder for the file password.

Property Value/Return Value

[Visual Basic] The file password reminder.

Remarks

This function goes directly to the restore file to get the password reminder, hence the user must have read access to the restore file.

ISSOConfigSS.RestoreSecret Method

The **RestoreSecret** method restores master secrets from the password protected backup file.

Syntax
VB

```
Sub RestoreSecret(
restoreFile as String,
filePassword as String);
```

C#

```
void RestoreSecret(
string restoreFile,
string filePassword);
```

Parameters

Parameter	Description
restoreFile	The file from which the secrets are to be restored.
filePasswordReminder	The password used to protect the restore file.

ISSOConfigStore Interface (COM)

The **ISSOConfigStore** interface provides administration functions for the Enterprise Single Sign-On (SSO) config store.

Requirements

Type Library: SSOConfigStore 1.0 Type Library (SSOConfigStore.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOConfigStore Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOConfigStore Members

The following table shows the ISSOConfigStore members.

Public Methods

Member	Description
DeleteConfigInfo	Delete the configuration information from the config store.
GetConfigInfo	Get the configuration information from the config store.
SetConfigInfo	Sets the configuration information in the config store.

See Also

Concepts

ISSOConfigStore Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOConfigStore Methods

The methods of the **ISSOConfigStore** interface are listed in the following table. For a complete list of **ISSOConfigStore** interface members, see ISSOConfigStore Members.

Public Methods

Method	Description
DeleteConfigInfo	Deletes the configuration information from the config store.
GetConfigInfo	Gets the configuration information from the config store.
SetConfigInfo	Sets the configuration information in the config store.

See Also

Concepts

ISSOConfigStore Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOConfigStore::DeleteConfigInfo

The **DeleteConfigInfo** method deletes the configuration information from the config store.

Syntax
C++

```
HRESULT DeleteConfigInfo(
BSTR bstrApplication,
BSTR bstrIdentifier,
);
```

VB

```
DeleteConfigInfo(
bstrApplication As BSTR,
bstrIdentifier As BSTR,
)
```

Parameters
bstrApplication

[in] String containing the external application name.

bstrApplication

[in] String containing the external application name.

bstrIdentifier

[in] String containing the identifier for the config info. This will typically be a GUID string.

bstrIdentifier

[in] String containing the identifier for the config info. This will typically be a GUID string.

Return Value

This method does not return a value.

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return code	Description
S_OK	The config info was successfully returned from the config store.
S_FALSE	The config info did not already exist.
E_ACCESSDENIED	Access denied.
E_INVALIDARG	Invalid argument.

Remarks

If the config info specified by the parameters does not already exist in the config store, this method will return S_FALSE.

If the *bstrSSOServer* parameter is NULL, the Single Sign-On (SSO) server location is obtained from the registry. If the server location is not available in the registry, the local computer is used.

Example

```
        ConfigStore

        bstrApplication
```

```
        bstrIdentifier
```

See Also

Concepts

ISSOConfigStore Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOConfigStore::GetConfigInfo

The **GetConfigInfo** method gets the configuration information from the config store.

Syntax
C++

```
HRESULT GetConfigInfo(
BSTR bstrApplication,
BSTR bstrIdentifier,
LONG lFlags,
IPropertyBag* ppbConfigInfo
);
```

VB

```
GetConfigInfo(
bstrApplication As BSTR,
bstrIdentifier As BSTR,
lFlags As LONG,
ppbConfigInfo As IPropertyBag
)
```

Parameters
bstrApplication

[in] String containing the Single Sign-On (SSO) server. This property is optional.

bstrApplication

[in] String containing the SSO server. This property is optional.

bstrIdentifier

[in] String containing the identifier for the config info. This string is typically a GUID string.

bstrIdentifier

[in] String containing the identifier for the config info. This string is typically a GUID string.

lFlags

[in] Long integer containing the flags.

lFlags

[in] Long integer containing the flags.

ppbConfigInfo

[in] Pointer to an empty property bag that is populated with the config info as name/value pairs.

ppbConfigInfo

[in] Pointer to an empty property bag that is populated with the config info as name/value pairs.

Return Value

This method does not return a value.

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return code	**Description**
S_OK	The config info was successfully returned from the config store.
E_ACCESSDENIED	Access denied.

E_INVALIDARG	Invalid argument.

Remarks

This method can be executed either in admin or run-time (lookup) mode. The caller specifies SSO_FLAG_LOOKUP if the run-time (lookup) mode is required. The default mode is admin mode.

In admin mode, masked properties are not returned. Instead, the property is missing. Unmasked properties are returned. Admin mode can specify any SSO server, not just the local computer.

In run-time mode, all properties, including masked properties, are returned. Because run-time mode only uses the SSO server on the current computer, the *bstrSSOServer* parameter will be ignored.

If the *bstrSSOServer* parameter is NULL, the SSO server location is obtained from the registry. (This applies to admin mode only. Run-time mode always uses the local computer.) If the server location is not available in the registry, the local computer is used.

To get the config info, this method is provided with an empty property bag that is populated with the properties. This allows the BizTalk Server 2006 implementation of the property bag to be used, which can handle the type conversion from BSTRs to the actual variant types based on a format convention specific to BizTalk Server 2006. The property values will be XML tags for Host Integration Server.

Example Code

```
        ConfigStore

        bstrApplication

        bstrIdentifier

        lFlags

        ppbConfigInfo
```

See Also

Concepts

ISSOConfigStore Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOConfigStore::SetConfigInfo

The **SetConfigInf**o method sets the configuration information in the config store.

Syntax
C++

```
HRESULT SetConfigInfo(
BSTR bstrApplication,
BSTR bstrIdentifier,
IPropertyBag* ppbConfigInfo
);
```

VB

```
SetConfigInfo(
bstrApplication As BSTR,
bstrIdentifier As BSTR,
ppbConfigInfo As IPropertyBag
)
```

Remarks
bstrApplication

[in] String containing the external application name.

bstrApplication

[in] String containing the external application name.

bstrIdentifier

[in] String containing the identifier for the config info. This string will typically be a GUID.

bstrIdentifier

[in] String containing the identifier for the config info. This string will typically be a GUID.

ppbConfigInfo

[in] Contains a pointer to a property bag containing the config info as name/value pairs.

ppbConfigInfo

[in] Contains a pointer to a property bag containing the config info as name/value pairs.

Return Values

This method does not return a value.

The method returns an **HRESULT**. Possible values include, but are not limited to, those in the following table.

Return code	Description
S_OK	The config info was successfully stored in the config store.
E_ACCESSDENIED	Access denied.
E_INVALIDARG	Invalid argument.

Remarks

This method can be used for originally creating the config info or for updating the config info.

If the config info does not already exist, all properties of the config info, as defined by the field info for the specified application, must be provided.

If the config info does already exist, a property within the config info can be missing. Only properties that are provided will be updated. At least one property must be provided.

Note that due to the use of the Single Sign-On (SSO) store, the first property cannot be masked.

If the ***bstrSSOServer*** parameter is NULL, the SSO server location is obtained from the registry. If not available in the registry, the local computer will be used.

Example

```
        ConfigStore

        bstrApplication

        bstrIdentifier

        ppbConfigInfo
```

See Also

Reference

ISSOConfigStore::GetConfigInfo

Concepts

ISSOConfigStore Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup1 Interface (COM)

The **ISSOLookup1** interface provides a lookup for user credentials associated with an application.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOLookup1 Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup1 Members

The following table shows the ISSOLookup1 member.

Public Methods

Member	Description
GetCredentials	Retrieves the user credentials for an application.

See Also

Concepts

ISSOLookup1 Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup1 Methods

The method of the **ISSOLookup1** interface is listed in the following table. For a complete list of **ISSOLookup1** interface members, see ISSOLookup1 Members.

Public Methods

Method	Description
GetCredentials	Retrieves the user credentials for an application.

See Also

Concepts

ISSOLookup1 Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup1.GetCredentials Method

The **GetCredentials** method retrieves the user credentials for an application.

Syntax
C++

```
HRESULT GetCredentials(
BSTR bstrApplicationName,
LONG lFlags,
BSTR* pbstrExternalUserName,
SAFEARRAY credentials
);
```

VB

```
Function GetCredentials(
bstrApplicationName As String,
lFlags As Long,
pbstrExternalUserName As String
)
As String
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

lFlags

[in] Long integer that specifies the flags to set. Using the SSO_FLAG_REFRESH indicates that the credential cache should be bypassed.

lFlags

[in] Long that specifies the flags to set. Using the SSO_FLAG_REFRESH indicates that the credential cache should be bypassed.

pbstrExternalUserName

[out] Pointer to a string that receives the external user name.

pbstrExternalUserName

[out] String that receives the external user name.

credentials

[out] String array that receives the credentials.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

String array that receives the credentials.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an Application User. You can only retrieve your own credential.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOLookup1 Interface (COM)

ISSOLookup1 Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup2 Interface (COM)

The **ISSOLookup2** interface provides a lookup for user credentials associated with an application.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOLookup2 Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup2 Members

The following table shows the ISSOLookup1 members.

Public Methods

Member	Description
GetCredentials	Retrieves the user credentials for an application.
LogonExternalUser	Logs on an external user.

See Also

Concepts

ISSOLookup2 Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup2 Methods

The methods of the **ISSOLookup2** interface are listed in the following table. For a complete list of **ISSOLookup2** interface members, see ISSOLookup2 Members.

Public Methods

Method	Description
GetCredentials	Retrieves the user credentials for an application.
LogonExternalUser	Logs on an external user.

See Also

Concepts

ISSOLookup2 Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup2.GetCredentials Method

The **GetCredentials** method retrieves the user credentials for an application.

Syntax
C++

```
HRESULT GetCredentials(
BSTR bstrApplicationName,
LONG lFlags,
BSTR* pbstrExternalUserName,
SAFEARRAY credentials
);
```

VB

```
Function GetCredentials(
bstrApplicationName As String,
lFlags As Long,
pbstrExternalUserName As String
)
As String
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

lFlags

[in] Long integer that specifies the flags to set. Using the SSO_FLAG_REFRESH indicates that the credential cache should be bypassed.

lFlags

[in] Long that specifies the flags to set. Using the SSO_FLAG_REFRESH indicates that the credential cache should be bypassed.

pbstrExternalUserName

[out] Pointer to a string that receives the external user name.

pbstrExternalUserName

[out] String that receives the external user name.

credentials

[out] String array that receives the credentials.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

String array that receives the credentials.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an Application User. You can only retrieve your own credential.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOLookup2 Interface (COM)

ISSOLookup2 Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOLookup2.LogonExternalUser Method

The **LogonExternalUser** method logs on an external user.

Syntax
C++

```
Int GetCredentials(
BSTR bstrApplicationName,
BSTR bstrUserName,
LONG lFlags,
Array ppsaCredentials
);
```

VB

```
Function GetCredentials(
bstrApplicationName As String,
bstrUserName As String,
lFlags As Long,
ppsaCredentials As String
)
As Integer
```

Parameters
bstrApplicationName

[in] String that specified the name of the application to log on to.

bstrApplicationName

[in] String that specified the name of the application to log on to.

bstrUserName

[in] String that specified the name of the external user to log on.

bstrUserName

[in] String that specified the name of the external user to log on.

lFlags

[in] Long integer that specifies the flags to set.

lFlags

[in] Long integer that specifies the flags to set.

ppsaCredentials

Array that holds the credentials of the user.

ppsaCredentials

Array that holds the credentials of the user.

Return Value

A windows handle.

A windows handle.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts

ISSOLookup2 Interface (COM)
ISSOLookup2 Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapper Interface (COM)

The **ISSOMapper** interface creates mappings between users and applications.

Requirements

Type Library: SSOMapper 1.0 Type Library (SSOMapper.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper Members

The following table shows the ISSOMapper members.

Public Methods

Member	Description
GetApplications	Retrieves the available applications for a Microsoft® Windows® user.
GetFieldInfo	Retrieves the field information for an application.
GetMappingsForExternalUser	Retrieves the mappings for an external user.
GetMappingsForWindowsUser	Retrieves the mappings for a Microsoft Windows user.
SetExternalCredentials	Stores a set of external credentials in the Single Sign-On (SSO) server database.
SetWindowsPassword	Sets the Microsoft Windows password. This method is not currently implemented.

See Also

Concepts

ISSOMapper Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper Methods

The methods of the **ISSOMapper** interface are listed in the following table. For a complete list of **ISSOMapper** interface members, see ISSOMapper Members.

Public Methods

Method	Description
GetApplications	Retrieves the available applications for a Microsoft® Windows® user.
GetFieldInfo	Retrieves the field information for an application.
GetMappingsForExternalUser	Retrieves the mappings for an external user.
GetMappingsForWindowsUser	Retrieves the mappings for a Microsoft Windows user.
SetExternalCredentials	Stores a set of external credentials in the Enterprise Single Sign-On (SSO) server database.
SetWindowsPassword	Sets the Microsoft Windows password. This method is not currently implemented.

See Also

Concepts

ISSOMapper Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.GetApplications Method

The **GetApplications** method retrieves the available applications for a Microsoft Windows user.

Syntax

C++

```
HRESULT GetApplications(
SAFEARRAY applications,
SAFEARRAY descriptions,
SAFEARRAY contactInfo
)
```

VB

```
Sub GetApplications(
applications As String,
descriptions As String,
contactInfo As String
)
```

Parameters

applications

[out] String array that returns the application name.

applications

[out] String array that returns the application name.

descriptions

[out] String array that returns a description for the application.

descriptions

[out] String array that returns a description for the application.

contactInfo

[out] String array that returns the contact information for the application.

contactInfo

[out] String array that returns the contact information for the application.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.GetFieldInfo Method

The **GetFieldInfo** method retrieves the field information for an application.

Syntax
C++

```
HRESULT GetFieldInfo(
BSTR bstrApplicationName,
SAFEARRAY labels,
SAFEARRAY flags
);
```

VB

```
Sub GetFieldInfo(
bstrApplicationName As String,
labels As String
flags As Long
)
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

labels

[out] String array that returns the field labels.

labels

[out] String array that returns the field labels.

flags

[out] Long integer array that returns the field flags.

flags

[out] Long array that returns the field flags.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	**Description**
S_OK	The method succeeded.

E_ACCESSDENIED	Access is denied to the caller.
E_FAIL	The application is disabled.
E_INVALIDARG	An invalid parameter was detected.

Remarks

The field information will only be returned if the application is currently enabled. An application cannot be enabled unless its field information is complete.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Interface (COM)

ISSOMapper Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.GetMappingsForExternalUser Method

The **GetMappingsForExternalUser** method retrieves the mappings for an external user.

Syntax
C++

```
HRESULT GetMappingsForExternalUser(
BSTR bstrApplicationName,
BSTR bstrExternalUserName,
SAFEARRAY mappings
);
```

VB

```
Function GetMappingsForExternalUser(
bstrApplicationName As String,
bstrExternalUserName As String)
As Variant
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrExternalUserName

[in] String that specifies the external user name. This value can contain spaces. If a NULL value is specified for this parameter, all mappings are returned for the specified application.

bstrExternalUserName

[in] String that specifies the external user name. This value can contain spaces. If a NULL value is specified for this parameter, all mappings are returned for the specified application.

mappings

[out] Object array that contains the mappings as **ISSOMapping** objects.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Variant array that contains the mappings as **ISSOMapping** objects.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.

E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Interface (COM)

ISSOMapper Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.GetMappingsForWindowsUser Method

The **GetMappingsForWindowsUser** method retrieves the mappings for a Microsoft Windows user.

Syntax

C++

```
HRESULT GetMappingsForWindowsUser(
BSTR bstrWindowsDomainName,
BSTR bstrWindowsUserName,
BSTR bstrApplicationName,
SAFEARRAY mappings
);
```

VB

```
Function GetMappingsForWindowsUser(
bstrWindowsDomainName As String,
bstrWindowsUserName As String,
bstrApplicationName As String
)
As Variant
```

Parameters

bstrWindowsDomainName

[in] String that specifies the Microsoft Windows domain name. This parameter is optional, but if it is specified, the *windowsUserName* parameter must also be specified. If this parameter is not specified, the current user context is used.

bstrWindowsDomainName

[in] String that specifies the Microsoft Windows domain name. This parameter is optional, but if it is specified, the *windowsUserName* parameter must also be specified. If this parameter is not specified, the current user context is used.

bstrWindowsUserName

[in] String that specifies the Microsoft Windows domain name. This parameter is optional, but if it is specified, the *windowsUserName* parameter must also be specified. If this parameter is not specified, the current user context is used.

bstrWindowsUserName

[in] String that specifies the Microsoft Windows domain name. This parameter is optional, but if it is specified, the *windowsUserName* parameter must also be specified. If this parameter is not specified, the current user context is used.

bstrApplicationName

[in] String that specifies the external application name. If this parameter is specified, only one mapping is returned if it exists. If this parameter is NULL, all mappings for the specified user are returned. This parameter is optional.

bstrApplicationName

[in] String that specifies the external application name. If this parameter is specified, only one mapping is returned if it exists. If this parameter is NULL, all mappings for the specified user are returned. This parameter is optional.

mappings

[out] Object array that contains the mappings as **ISSOMapping** objects.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Variant array that contains the mappings as **ISSOMapping** objects.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

An Application Administrator must specify the *bstrApplicationName* value.

Users described as Application User can only access their own mappings. They cannot access the mappings for another user.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Interface (COM)

ISSOMapper Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.SetExternalCredentials Method

The **SetExternalCredentials** method stores a set of external credentials in the Enterprise Single Sign-On (SSO) server database.

Syntax
C++

```
HRESULT SetExternalCredentials(
BSTR bstrApplicationName,
BSTR bstrExternalUserName,
SAFEARRAY externalCredentials
);
```

```
[Visual Basic]
Sub SetExternalCredentials(
bstrApplicationName As String,
bstrExternalUserName As String,
externalCredentials As String
)
```

Parameters
bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrExternalUserName

[in] String that specifies the external user name.

bstrExternalUserName

[in] String that specifies the external user name.

externalCredentials

[in] String array that specifies the external credentials to be stored.

externalCredentials

[in] String array that specifies the external credentials to be stored.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.

E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

Users described as Application User can only set their own credentials. Also, the number of external credentials provided must match the number of fields expected by the external application.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapper Interface (COM)

ISSOMapper Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapper.SetWindowsPassword Method

The **SetWindowsPassword** method sets the Microsoft Windows password. This method is not currently implemented.

Syntax
C++

```
HRESULT SetWindowsPassword(
BSTR bstrWindowsPassword
);
```

VB

```
Sub SetWindowsPassword(
bstrWindowsPassword As String
)
```

Parameters
bstrWindowsPassword

[in] String that specifies the Windows password.

bstrWindowsPassword

[in] String that specifies the Windows password.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
E_NOTIMPL	The method is not implemented.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapper Interface (COM)
ISSOMapper Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapper2 Interface (COM)

The **ISSOMapper2** interface creates mappings between users and applications

Requirements

Type Library: SSOMapper 1.0 Type Library (SSOMapper.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

ISSOMapper2 Members

The following table shows the ISSOMapper2 methods.

Public Methods

ISSOMapper2.GetApplications2 Method	Retrieves the available applications for a Microsoft® Windows® user.
ISSOMapper.GetFieldInfo Method	Retrieves the field information for an application.
ISSOMapper.GetMappingsForExternalUser Method	Retrieves the mappings for an external user.
ISSOMapper.GetMappingsForWindowsUser Method	Retrieves the mappings for a Microsoft Windows user.
ISSOMapper.SetExternalCredentials Method	Stores a set of external credentials in the Single Sign-On database.
ISSOMapper.SetWindowsPassword Method	Sets the Microsoft Windows password. This method is not currently implemented.

ISSOMapper2 Methods

The following table shows the **ISSOMapper2** methods.

Public Methods

Method	Description
ISSOMapper2.GetApplications2 Method	Retrieves the available applications for a Microsoft® Windows® user.

ISSOMapper2.GetApplications2 Method

The GetApplications2 method retrieves the available applications for a Microsoft Windows user.

Syntax
VB

```
Sub GetApplications2(
applications As String,
descriptions As String,
contactInfo As String,
userAccounts As String,
adminAccounts As String,
flags As Long
);
```

C#

```
HRESULT GetApplications2(
SAFEARRAY applications,
SAFEARRAY descriptions,
SAFEARRAY contactInfo,
SAFEARRAY userAccounts,
SAFEARRAY adminAccounts,
SAFEARRAY flags
);
```

Parameters

Parameter	**Description**
applications	String array that returns the application name.
descriptions	String array that returns a description of the application.
contactInfo	String array that returns the contact information for the application.
userAccounts	String array that returns all user accounts for the application. GetApplications2 returns a null in this parameter when called by an application user.
adminAccounts	String array that returns all administration accounts for the application. GetApplications2 returns a null in this parameter when called by an application user.
flags	Integer array that returns the flags for the application.

Property Value/Return Value

[C++] This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

[Visual Basic] Not Applicable.

Exceptions

[C++] This method returns an HRESULT containing one of the values in the following table.

[Visual Basic] This method indicates errors by setting the Number property of the global Err object to one of the values in the following table.

Value	**Description**

S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDREG	An invalid parameter was detected.

ISSOMapping Interface (COM)

The **ISSOMapping** interface manages the state of a mapping instance.

Requirements

Type Library: SSOMapper 1.0 Type Library (SSOMapper.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapping Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping Members

The following tables show the **ISSOMapper** members.

Public Properties

Property	Description
ApplicationName	Specifies the name of the application.
ExternalUserName	Specifies the external (non-Microsoft® Windows®) user name.
Flags	Reserved for internal use only.
WindowsDomainName	Specifies the Microsoft Windows domain name.
WindowsUserName	Specifies the Microsoft Windows user name.

Public Methods

Method	Description
Create	Creates the mapping.
Delete	Deletes the mapping.
Disable	Disables the mapping.
Enable	Enables the mapping.

See Also

Concepts

ISSOMapping Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping Properties

The properties of the **ISSOMapping** interface are listed in the following table. For a complete list of **ISSOMapping** interface members, see ISSOMapping Members.

Public Properties

Property	Description
ApplicationName	Specifies the name of the application.
ExternalUserName	Specifies the external (non-Microsoft® Windows®) user name.
Flags	Reserved for internal use only.
WindowsDomainName	Specifies the Microsoft Windows domain name.
WindowsUserName	Specifies the Microsoft Windows user name.

See Also

Concepts

ISSOMapping Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping.ApplicationName Property

The **ApplicationName** property specifies the name of the application.

Syntax
C++

```
HRESULT ISSOMapping::get_ApplicationName(
BSTR* ApplicationName
);
HRESULT ISSOMapping::put_ApplicationName(
BSTR ApplicationName
);
```

VB

```
Property ApplicationName() As String
```

Parameters
ApplicationName

[in] When putting the property, a string that contains the application name. [out, retval] When getting the property, a string used to return the application name.

None.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping.ExternalUserName Property

The **ExternalUserName** property specifies the external (non-Microsoft Windows) user name.

Syntax
C++

```
HRESULT ISSOMapping::get_ExternalUserName(BSTR* ExternalUserName);
HRESULT ISSOMapping::put_ExternalUserName(BSTR ExternalUserName);
```

VB

```
Property ExternalUserName() As String
```

Parameters
ExternalUserName

[in] When putting the property, a string that contains the external user name. [out, retval] When getting the property, a string used to return the external user name.

None.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping.Flags Property

The **Flags** property is reserved for internal use only.

Syntax
C++

```
HRESULT ISSOMapping::get_Flags(BSTR* Flags);
HRESULT ISSOMapping::put_Flags(BSTR Flags);
```

VB

```
Property Flags() As String
```

Parameters
Flags

[in] When putting the property, a string that contains the flag value. [out, retval] When getting the property, a string used to return the flag value.

None.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping.WindowsDomainName Property

The **WindowsDomainName** property specifies the Microsoft Windows domain name.

Syntax

C++

```
HRESULT ISSOMapping::get_WindowsDomainName(BSTR* WindowsDomainName);
HRESULT ISSOMapping::put_WindowsDomainName(BSTR WindowsDomainName);
```

VB

```
Property WindowsDomainName() As String
```

Parameters

WindowsDomainName

[in] When putting the property, a string that contains the Windows domain name. Maximum length is 15 characters.

[out, retval] When getting the property, a string used to return the Windows domain name.

None.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapping Interface (COM)

ISSOMapping Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping.WindowsUserName Property

The **WindowsUserName** property specifies the Microsoft Windows user name.

Get method:

Syntax
C++

```
HRESULT ISSOMapping::get_WindowsUserName(BSTR*WindowsUserName);
HRESULT ISSOMapping::put_WindowsUserName(BSTR WindowsUserName);
```

VB

```
Property WindowsUserName() As String
```

Parameters
WindowsUserName

[in] When putting the property, a string that contains the Windows user name. [out, retval] When getting the property, a string used to return the Windows user name.

None.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping Methods

The methods of the **ISSOMapping** interface are listed in the following table. For a complete list of **ISSOMapping** interface members, see ISSOMapping Members.

Public Methods

Method	Description
Create	Creates the mapping.
Delete	Deletes the mapping.
Disable	Disables the mapping.
Enable	Enables the mapping.

See Also

Concepts

ISSOMapping Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping.Create Method

The **Create** method creates the mapping.

Syntax
C++

```
HRESULT Create(
LONG lFlags
);
```

VB

```
Sub Create(
lFlags As Long
)
```

Parameters
lFlags

For the current release, *lFlags* can be set to SSO_FLAG_REFRESH only.

lFlags

For the current release, *lFlags* can be set to SSO_FLAG_REFRESH only.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping.Delete Method

The **Delete** method deletes the mapping.

Syntax

C++

```
HRESULT Delete();
```

VB

```
Sub Delete()
```

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

After the mapping has been deleted, the mapping object can no longer be used and should be released.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOMapping Interface (COM)

ISSOMapping Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOMapping.Disable Method

The **Disable** method disables the mapping.

Syntax
C++

```
HRESULT Disable();
```

VB

```
Sub Disable()
```

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOMapping.Enable Method

The **Enable** method enables the mapping.

Syntax
C++

```
HRESULT Enable(
LONG lFlags
);
```

VB

```
Sub Enable(
lFlags As Long
)
```

Parameters
lFlags

Not currently implemented.

lFlags

Not currently implemented.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Not applicable.

Error Values

This method returns an HRESULT containing one of the values in the following table.

This method indicates errors by setting the **Number** property of the global **Err** object to one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOMapping Interface (COM)
ISSOMapping Members
Other Resources
Programming with Enterprise Single Sign-On

ISSONotification Interface (COM)

The **ISSONotification** interface handles password changes to and from the client and server.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSONotification Members

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification Members

The following table shows the **ISSONotification** members.

Public Methods

Member	Description
InitializeAdapter	Initializes the password sync adapter to the ENTSSO system.
SendNotification	Sends a notification, such as a password change, from the adapter to the ENTSSO system.
ReceiveNotification	Receives a notification from the ENTSSO system, such as password changes.
ShutdownAdapter	Indicates that the password sync adapter is shutting down.

See Also

Concepts

ISSONotification Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification Methods

The method of the **ISSONotification** interface is listed in the following table. For a complete list of **ISSONotification** interface members, see ISSONotification Members.

Public Methods

Member	Description
InitializeAdapter	Initializes the password sync adapter to the ENTSSO system.
SendNotification	Sends a notification, such as a password change, from the adapter to the ENTSSO system.
ReceiveNotification	Receives a notification from the ENTSSO system, such as password changes.
ShutdownAdapter	Indicates that the password sync adapter is shutting down.

See Also

Concepts

ISSONotification Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification.InitializeAdapter Method

Initializes the password sync adapter to the ENTSSO system.

Syntax
C++

```
HRESULT InitializeAdapter(
BSTR strAdapterName,
ULONG lFlags,
ULONGLONG* hNotifyEvent,
GUID* guidTrackingId);
```

Parameters
bstrAdapterName

[in] The unique adapter name.

ulFlags

[in] A bitwise combination of the SSO_NOTIFICATION_FLAG values.

phNotifyEvent

[out] When this method returns, contains an event handle created by PS Helper. You should cast the return value to a HANDLE on return, as MIDL does not support the HANDLE data type. This parameter can be NULL if the event handle is not required by the adapter.

pguidTrackingId

[out] When this method returns, contains the tracking ID generated by ENTSSO. The tracking ID is used for auditing purposes. This parameter can be NULL if the tracking ID is not required by the adapter.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The initialization was successful.
S_FALSE	The initialization was successful, but was a reconnect. For more information, see the Remarks section.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.

Remarks

Before calling **InitializeAdapter**, you must have entered the relevant adapter name into ENTSSO.

InitializeAdapter should be the first method your adapter calls, because you cannot call any other **ISSONotification** methods before you call **InitializeAdapter**. You should not call **InitializeAdapter** again until after you call **ShutdownAdapter**. Once you shut down the adapter, however, you may call **InitializeAdapter** at any time to reconnect.

InitializeAdapter initiates communication between the PS Helper and the adapter. When your adapter calls **InitializeAdapter**, the PS Helper calls the Enterprise Single Sign-On (ENTSSO) service over encrypted LRPC. Using the adapter name, the ENTSSO service grants or denies access based on the access account that was defined for the current adapter.

You must have started the ENTSSO service before calling **InitializeAdapter**. PS Helper cannot automatically start ENTSSO because the adapter process might not be running with sufficient privileges to start a service. Therefore, your adapter must initiate all communication between the adapter and ENTSSO.

PS Helper first establishes a connection with the ENTSSO service, and then creates a named event. PS Helper then passes the named event to the ENTSSO service. ENTSSO uses the event signal to the PS Helper when a notification arrives for the adapter. ENTSSO returns the event to the adapter so that the adapter can wait on the event, or else ignore the event and allow PS Helper to wait instead. This gives more flexibility to the adapter for the adapter threading model. The event is valid for the adapter until ENTSSO completes the processing initiated by a call to **ShutdownAdapter**.

The ENTSSO service accepts the initial remote procedure call (RPC) call from the PS Helper if ENTSSO is running. ENTSSO then performs an access check. If ENTSSO is unable to access the database to obtain the adapter configuration information, then ENTSSO returns an E_ACCESSDENIED event. If ENTSSO has the adapter configuration information, but cannot currently contact the database, then ENTSSO continues to accept password change notifications and buffers the notifications locally and encrypt the notifications in a local temporary file.

InitializeAdapter also returns E_ACCESSDENIED if the adapter is deleted or disabled.

For all errors, more detailed information will be available in the Windows Event Log.

It is assumed that the adapter knows the appropriate name to use when communicating with ENTSSO.

InitializeAdapter is single-threaded. All other threads calling **InitializeAdapter** are blocked until **InitializeAdapter** has completed. It is also synchronized with the **ShutdownAdapter** method.

It is possible that the adapter process terminates before you can issue a **ShutdownAdapter**. In this case, and if ENTSSO receives another **InitializeAdapter** before ENTSSO receives a corresponding **ShutdownAdapter**, ENTSSO treats the second **ShutdownAdapter** call as a reconnect. In this case, ENTSSO cleans up and invalidates the existing event, and creates a new handle. ENTSSO also completes any pending **ReceiveNotifications** for the old event handle with a shutdown notification.

In the case of a reconnect, a new tracking ID is returned from the **InitializeAdapter**. ENTSSO returns a new tracking ID because the tracking ID returned from **InitializeAdapter** can be considered a session ID. Further, ENTSSO reissues any pending (but unconfirmed) notifications to the adapter.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSONotification Interface (COM)

ISSONotification Members

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification.SendNotification Method

Sends a notification, such as a password change, from the adapter to the ENTSSO system.

Syntax
C++

```
HRESULT SendNotification(
SSendNotification SendNotification,
GUID* pguidTrackingId
);
```

Parameters
SendNotification

[in] The notification to send to ENTSSO from the adapter.

pguidTrackingId

[out] When this method returns, contains the tracking ID generated by ENTSSO. You can use the tracking ID for auditing purposes or to correlate request responses. May be NULL.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The method was successful.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.
ENTSSO_E_WRONG_STATE	This method has been called in the wrong state.
ENTSSO_E_INVALID_NOTIFICATION	Invalid notification type.

Remarks

You can use **SendNotification** to send password changes and other notifications to the ENTSSO system.

If **SendNotification** returns S_OK, this does not mean that a password change was completed on the destination system. Instead, receiving an S_OK means ENTSSO has accepted and eventually will complete your request.

A password change from an external system can have several consequences:

- If a partial password sync is configured, then the SSO database might be updated, if a current mapping exists for the external account.
- If a full password sync is configured, then the password change might also be made to a Windows account.

If the external account has no current mapping in the SSO database, the password change might have no effect.

The password change complete notification is issued when the password change is considered complete from the ENTSSO point-of-view, which as discussed above, could mean different things. In some cases, it could mean no change was done, that only the SSO database was updated, or that the Windows password was changed.

Note that password change complete notifications sent back to the adapter are not completely reliable. Under some error conditions, Single Sign-On may never actually receives the requested notifications.

In ENTSSO, the definition of credentials, such as those sent by **SendNotification** for password updates, is more flexible than a simple password. When you define an SSO application, you also define the credential fields. The fields identify the labels to use for the UI fields, and whether those fields are masked or not. In addition, there is also a special flage which specifies whether the field should be synchronized or not. Field 0 is a special case and defines the label for the user ID. For more information, see the ISSOAdmin Interface.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSONotification Interface (COM)

ISSONotification Members

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification.ReceiveNotification Method

Receives a notification from the ENTSSO system, such as password changes.

Syntax
C++

```
HRESULT ReceiveNotification(
ULONG ulNotificationFlagsIn,
SReceiveNotification* pReceiveNotification,
GUID* pguidTrackingId
);
```

Parameters
bstrAdapterName

[in] The unique adapter name.

ulNotificationFlagsIn

[in] The notification flags to control this notification, from the SSO_NOTIFICATION_FLAG enumeration.

pReceiveNotification

[in] Pointer for the received notification.

pguidTrackingId

[out] The tracking ID. The ENTSSO system will generate a tracking ID and return it to the caller. The tracking ID is used for auditing purposes and can also be used by the adapter to correlate responses to requests. This parameter can be NULL if the tracking ID is not required by the adapter.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The method was successful.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.
ENTSSO_E_NO_NOTIFICATIONS	There are no notifications to be received.
ENTSSO_E_WRONG_STATE	This method has been called in the wrong state.

Remarks

You can use **ReceiveNotification** to receive both password changes and other notifications from the ENTSSO system.

You may call **ReceiveNotification** with or without a WAIT flag. If you specify the WAIT flag, **ReceiveNotification** blocks until a notification is available. Doing so enables you to determine if you want to dedicate a thread for receiving notifications from the ENTSSO service, or whether you want to use the event handle returned from **InitializeAdapter** to perform your own waits, shared with other events.

If **ReceiveNotification** is waiting when you call ShutdownAdapter, then **ReceiveNotification** returns a SHUTDOWN notification as the last notification. The SHUTDOWN notification preempts any other pending notifications.

It is possible that multiple threads could be calling **ReceiveNotification** for the same adapter name. In this case, the request is single-threaded at the ENTSSO service, and only one **ReceiveNotification** completes with valid information. The threads

complete with either the next notification or NONE. The reason is that each of these threads waits for the same event.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSONotification Interface (COM)

ISSONotification Members

Other Resources

Programming with Enterprise Single Sign-On

ISSONotification.ShutdownAdapter Method

Indicates that the password sync adapter is shutting down.

Syntax
C++

```
HRESULT ShutDownAdapter(
GUID* pguidTrackingId
);
```

Parameters
pguidTrackingId

[out] When this method returns, contains the tracking ID. The tracking ID is the same tracking ID that ENTSSO returns in the initialization process, which you can use for auditing purposes. May be NULL.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The shutdown was successful.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.
ENTSSO_E_WRONG_STATE	This method has been called in the wrong state.

Remarks

ShutdownAdapter should be the last method you call. You may call neither SendNotification nor ReceiveNotification after you call **ShutdownAdapter.** The only method you may call afterward is InitializeAdapter, which initializes a new session.

Calls to **SendNotification** or **ReceiveNotification** that are in progress (on other threads) when you call **ShutdownAdapter** may receive ENTSSO_E_WRONG_STATE, although one thread calling **ReceiveNotification** receives the SHUTDOWN_COMPLETE notification.

ShutdownAdapter is single-threaded. ENTSSO blocks all other threads calling **ShutdownAdapter** until **ShutdownAdapter** has completed. **ShutdownAdapter** is also synchronized with the **InitializeAdapter** method.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSONotification Interface (COM)
ISSONotification Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOTicket Interface (COM)

The **ISSOTicket** interface issues and redeems Enterprise Single Sign-On (SSO) server tickets.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOTicket Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOTicket Members

The following table shows the **ISSOTicket** members.

Public Methods

Member	Description
IssueTicket	Issues an Enterprise Single Sign-On (SSO) server ticket for authenticating a user on an application.
RedeemTicket	Redeems an SSO server ticket that was previously issued with the **IssueTicket** method.

See Also

Concepts

ISSOTicket Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOTicket Methods

The methods of the **ISSOTicket** interface are listed in the following table. For a complete list of **ISSOTicket** interface members, see ISSOTicket Members.

Public Methods

Method	Description
IssueTicket	Issues an Enterprise Single Sign-On (SSO) server ticket for authenticating a user on an application.
RedeemTicket	Redeems an SSO server ticket that was previously issued with the **IssueTicket** method.

See Also

Concepts

ISSOTicket Interface (COM)

Other Resources

Programming with Enterprise Single Sign-On

ISSOTicket.IssueTicket Method

The **IssueTicket** method issues an Enterprise Single Sign-On (SSO) server ticket for authenticating a user on an application.

Syntax
C++

```
HRESULT IssueTicket(
LONG lFlags,
BSTR* pbstrTicket
);
```

Parameters
lFlags

[in] This parameter is ignored in the current release.

lFlags

[in] This parameter is ignored in the current release.

pbstrTicket

[out] Pointer to a string that receives the ticket. This is a Base64-encoded value for use in XML documents.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

To access this method, you must be an Application User. You can only issue a ticket for yourself.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
ISSOTicket Interface (COM)
ISSOTicket Members
Other Resources
Programming with Enterprise Single Sign-On

ISSOTicket.RedeemTicket Method

The **RedeemTicket** method redeems an Enterprise Single Sign-On (SSO) server ticket that was previously issued with the **IssueTicket** method.

Syntax

C++

```
HRESULT RedeemTicket(
BSTR bstrApplicationName,
BSTR bstrTicket,
LONG lFlags,
BSTR* pbstrExternalUserName,
SAFEARRAY BSTR
);
```

Parameters

bstrApplicationName

[in] String that specifies the application name. This parameter cannot be NULL, an empty string, or contain spaces. Application names are not case-sensitive, but case will be preserved. For example, ABC, abc, and AbC are considered to be the same application.

bstrTicket

[in] String that specifies the ticket value obtained from the **IssueTicket** method.

lFlags

[in] Long integer that specifies the flags to set. Use the flag SSO_FLAG_REFRESH to indicate that the credential cache should be bypassed.

pbstrExternalUserName

[out] Pointer to a string that receives the external user name associated with the ticket.

BSTR

[out] String that receives the external credentials associated with the ticket. If there are no credentials, the size of the returned array is zero.

Return Value

This method returns an HRESULT indicating whether it completed successfully. For more details, see the Error Values section.

Error Values

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The method succeeded.
E_ACCESSDENIED	Access is denied to the caller.
E_INVALIDARG	An invalid parameter was detected.

Remarks

Because the credentials are returned in plain text by this method, the caller should be careful to clear (overwrite) them as soon as possible after use.

To access this method, you must be an SSO Administrator, SSO Affiliate Administrator, or an Application Administrator.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

ISSOTicket Interface (COM)

ISSOTicket Members

Other Resources

Programming with Enterprise Single Sign-On

ISSOPSWrapper Interface (COM)

The ISSOPSWrapper interface allows developers to create password sync adapters in either managed or native code.

Requirements

Type Library: SSOPSHelper 1.0 Type Library (SSOPSHelper.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

Remarks

ISSOPSWrapper implements the same methods as ISSONotification but does not use structures.

See Also

Other Resources

Programming with Enterprise Single Sign-On

ISSOPSWrapper Members

The following table shows the ISSOPSWraper members.

Public Methods

ISSOWrapper.InitializeAdapter Method

Initializes the adapter.

ISSOWrapper.ReceiveNotification Method

Receives a notification to the adapter from the ENTSSO service.

ISSOWrapper.SendNotification Method

Sends a notification from the adapter to the ENTSSO service.

ISSOWrapper.ShutdownAdapter Method

Indicates to the ENTSSO service that the adapter is shutting down.

ISSOWrapper Methods

The following table describes the ISSOWrapper methods.

Public Methods

ISSOWrapper.InitializeAdapter Method

Initializes the adapter.

ISSOWrapper.ReceiveNotification Method

Receives a notification to the adapter from the ENTSSO service.

ISSOWrapper.SendNotification Method

Sends a notification from the adapter to the ENTSSO service.

ISSOWrapper.ShutdownAdapter Method

Indicates to the ENTSSO service that the adapter is shutting down.

ISSOWrapper.InitializeAdapter Method

Initializes the password sync adapter to the ENTSSO system.

Syntax
C++

```
HRESULT InitializeAdapter(
BSTR bstrAdapterName,
ULONG ulFlags,
ULONGLONG* phNotifyEvent,
GUID* pguidTrackingId
);
```

Parameters
bstrAdapterName

The unique adapter name.

ulFlags

A bitwise combination of the SSO_NOTIFICATION_FLAG values.

phNotifyEvent

When this method returns, contains an event handle created by PS Helper. You should cast the return value to a HANDLE on return, as MIDL does not support the HANDLE data type. This parameter can be NULL if the event handle is not required by the adapter.

pguidTrackingId

When this method returns, contains the tracking ID generated by ENTSSO. The tracking ID is used for auditing purposes. This parameter can be NULL if the tracking ID is not required by the adapter.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Exceptions section.

Exceptions

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The initialization was successful.
S_FALSE	The initialization was successful, but was a reconnect. For more information, see the Remarks section.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.

Remarks

InitializeAdapter acts in the same way as the ISSONotification.InitializeAdapter Method.

ISSOWrapper.SendNotification Method

Sends a notification from the adapter to the ENTSSO service.

Syntax
C++

```
HRESULT SendNotification(
ULONG ulNotificationType,
ULONG ulNotificationFlags,
Guid guidTrackingIdIn,
bstr bstrExternalAccount,
bstr bstrNewExternalPassword,
bstr bstrOldExternalPassword,
ulonglong ullTimestamp,
ulonglong ullErrorCode,
bstr bstrErrorMessage,
out Guid *pguidTrackingIdOut
);
```

Parameters

Parameter	Description
ulNotificationType	Notification type.
ulNotificationFlags	Notification flags.
guidTrackingIdIn	The tracking id of the password change to be confirmed.
bstrExternalAccount	The external account for which the password has changed.
bstrNewExternalPassword	The new password for the external account.
bstrOldExternalPassword	Optional. The old password for the external account.
ullTimestamp	The timestamp when the password change was made, or zero to use the current time.
ullErrorCode	If non-zero, the external password change failed. errorCode will be written to the event log. Zero for successful external password changes.
bstrErrorMessage	Optional. The message to write to the event log if the password change failed.
pguidTrackingIdOut	Optional. On return, contains a pointer to a GUID to receive the tracking id.

Property Value/Return Value
E_ACCESS_DENIED

Access is denied.

Exceptions

Remarks

SendNotification uses different parameters depending on the notificationType. The following table describes the necessary parameters for each type of notification.

notificationType	Parameters

SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE	externalAccount newExternalPassword oldExternalPassword timestamp
SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE_COMPLETE	trackingIdIn errorCode errorMessage
SSO_NOTIFICATION_TYPE_STATUS_OFFLINE	errorCode errorMessage
SSO_NOTIFICATION_TYPE_PASSWORD_EXPIRED	externalAccount
SSO_NOTIFICATION_TYPE_STATUS_REQUEST	None
SSO_NOTIFICATION_TYPE_STATUS_ONLINE	None

Requirements

ISSOWrapper.ReceiveNotification Method

Receives a notification to the adapter from the ENTSSO service.

Syntax
C++

```
void ReceiveNotification(
ULONG ulNotificationFlagsIn,
GUID guidTrackingIdIn,
out ULONG *pulNotificationType,
out ULONG *pulNotificationFlagsOut,
out BSTR *pbstrExternalAccount,
out BSTR *pbstrNewExternalPassword,
out BSTR *pbstrOldExternalPassword,
out ULONGLONG *pullTimestamp,
out ULONGLONG *pullErrorCode,
out BSTR *pbstrErrorMessage,
out SAFEARRAY(BSTR) *ppsaAdapters,
out BSTR *pbstrAdapterName,
out GUID *pguidTrackingIdOut
);
```

Parameters

Parameter	Description
ulNotificationFlagsIn	The notification flags. Specify SSO_NOTIFICATION_FLAG_WAIT if you want your call to block waiting for a notification, or SSO_NOTIFICATION_FLAG_NONE.
guidTrackingIdIn	The tracking ID.
pulNnotificationType	On return, contains a pointer to a ULONG that will receive the notification type.
pulNotificationFlagsOut	On return, contains a pointer to a ULONG that will receive the notification flags
pbstrExternalAccount	On return, contains the external account for which the password should be changed.
newExternalPassword	On return, contains the new password for the external account.
pbstrOldExternalPassword	Optional. On return, contains the old password for the external account.
pullTimestamp	On return, contains the timestamp when the password change was made.
pullErrorCode	On return, contains the error code.
pbstrErrorMessage	On return, contains an error message.
ppsaAdapters	On return, contains an array of adapter names.
pbstrAdapterName	On return, contains the adapter name.

pbuidTrackingIdOut	On return, contains the tracking ID.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Exceptions section.

Exceptions

E_ACCESS_DENIED

Access is denied.

Remarks

ReceiveNotification uses different parameters to return information depending on the notificationType. The following table describes the relationship between the different parameters and types of notifications.

notificationType	**Parameters**
SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE	externalAccount newExternalPassword oldExternalPassword timestamp
SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE_COMPLETE	trackingIdIn errorCode errorMessage
SSO_NOTIFICATION_TYPE_STATUS_OFFLINE	errorCode errorMessage
SSO_NOTIFICATION_TYPE_PASSWORD_EXPIRED	externalAccount
SSO_NOTIFICATION_TYPE_STATUS_REQUEST	None
SSO_NOTIFICATION_TYPE_STATUS_ONLINE	None

ISSOWrapper.ShutdownAdapter Method

Indicates to the ENTSSO service that the adapter is shutting down.

Syntax
C++

```
HRESULT ShutDownAdapter(
GUID* pguidTrackingId
);
```

Parameters

Parameter	Description
pguidtrackingId	When this method returns, contains the tracking ID. The tracking ID is the same tracking ID that ENTSSO returns in the initialization process, which you can use for auditing purposes. May be NULL.

Return Value

This method returns an HRESULT indicating whether it completed correctly. For more information, see the Error Values section.

Exceptions

This method returns an HRESULT containing one of the values in the following table.

Value	Description
S_OK	The shutdown was successful.
E_ACCESSDENIED	Access is denied.
ENTSSO_E_NO_SERVER	Could not contact the ENTSSO server. Check that the ENTSSO service is running.
ENTSSO_E_WRONG_STATE	This method has been called in the wrong state.

Remarks

Example

This is the description for a Code Example.

Optional comments.

.NET Framework Equivalent

Optional .NET Framework equivalent section.

Requirements
Subhead

SAdapter Structure (COM)

Describes an adapter.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

SAdapter Members

Other Resources

Programming with Enterprise Single Sign-On

SAdapter Members

The following table shows the **SAdapter** members.

Public Fields

Field	Description
bstrAdapterName	String containing the adapter name.

See Also

Concepts

SAdapter Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SAdapter Fields

The fields of the **SAdapter** interface is listed in the following table. For a complete list of **SExternal** interface members, see SAdapter Members.

Public Fields

Field	Description
bstrAdapterName	String containing the adapter name.

See Also

Concepts

SAdapter Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SAdapter.bstrAdapterName Field

String containing the adapter name.

Remarks

You must free the string after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

SAdapter Structure (COM)

SAdapter Members

Other Resources

Programming with Enterprise Single Sign-On

SAdapterInGroup Structure (COM)

Describes the names of one or more adapters in a group.

Syntax

```
public: __value struct SAdapterInGroup
```

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

SAdapterInGroup Members

Other Resources

Creating a Single Sign-On Application

SAdapterInGroup Members

The following table shows the **SAdapterInGroup** members.

Public Fields

Field	Description
psaAdapters	Pointer to a SafeArray of adapter names in the group.

See Also

Reference

SAdapterInGroup Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SAdapterInGroup Fields

The fields of the **SAdapterInGroup** interface are listed in the following table. For a complete list of **SExternal** interface members, see SAdapterInGroup Members.

Public Fields

Field	Description
psaAdapters	Pointer to a SafeArray of adapter names in the group.

See Also

Reference

SAdapterInGroup Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SAdapterInGroup.psaAdapters Field

Pointer to a SafeArray of adapter names in the group.

Remarks

This **SafeArray** must be destroyed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Reference

SAdapterInGroup Structure (COM)

Concepts

SAdapterInGroup Members

Other Resources

Programming with Enterprise Single Sign-On

SExternalAccount Structure (COM)

Describes an external account.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

SExternalAccount Members

Other Resources

Programming with Enterprise Single Sign-On

SExternalAccount Members

The following table shows the **SExternal** members.

Public Fields

Field	Description
bstrExternalAccount	A string describing the external account.

See Also

Other Resources

Programming with Enterprise Single Sign-On

SExternalAccount Fields

The fields of the **SExternalAccount** interface are listed in the following table. For a complete list of **SExternal** interface members, see SExternalAccount Members.

Public Fields

Field	Description
bstrExternalAccount	A string describing the external account.

See Also

Concepts

SExternalAccount Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SExternalAccount.bstrExternalAccount Field

A string describing the external account.

Syntax
C++

```
public: BSTR bstrExternalAccount;
```

Remarks

BSTR must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Concepts
SExternalAccount Structure (COM)
SExternalAccount Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChange Structure (COM)

Describes change to a password.

Syntax

```
public: __value struct SPasswordChange
```

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Concepts

SPasswordChange Members

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChange Members

The following table shows the **SPasswordChange** members.

Public Fields

Field	Description
bstrExternalAccount	A string describing the external account.
psaNewExternalPassword	Pointer to a SafeArray of new external credentials.
psaOldExternalPassword	Pointer to a SafeArray of new external credentials.
ullTimestamp	An integer containing a UTC timestamp in FILETIME format.

See Also

Reference

SPasswordChange Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChange Fields

The fields of the **SPasswordChange** interface are listed in the following table. For a complete list of **SExternal** interface members, see SPasswordChange Members

Public Fields

Field	Description
bstrExternalAccount	A string describing the external account.
psaNewExternalPassword	A string containing the new external credentials.
psaOldExternalPassword	A string containing the new external credentials.
ullTimeStamp	An integer containing a UTC timestamp in FILETIME format.

See Also

Reference

SPasswordChange Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChange.bstrExternalAccount Field

A string describing the external account.

Syntax
C++

```
public: BSTR bstrExternalAccount;
```

Remarks

BSTR must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChange Structure (COM)
Concepts
SPasswordChange Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChange.psaNewExternalPassword Field

A string containing the new external credentials.

Syntax
C++

```
public: BSTR psaNewExternalCredentials;
```

Remarks

Must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChange Structure (COM)
Concepts
SPasswordChange Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChange.psaOldExternalPassword Field

A string containing the new external credentials.

Syntax
C++

```
public: BSTR psaOldExternalCredentials;
```

Remarks

Can be NULL. Must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChange Structure (COM)
Concepts
SPasswordChange Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChange.ullTimeStamp Field

An integer containing a UTC timestamp in FILETIME format.

Syntax
C++

```
public: ULONGLONG ullTimeStamp;
```

Remarks

If zero, then Enterprise Single Sign-On (ENTSSO) uses the current time.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChange Structure (COM)
Concepts
SPasswordChange Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChangeComplete Structure (COM)

Describes when a change to a password has been completed.

Syntax

```
public: __value struct SPasswordChangeComplete
```

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Reference

SPasswordChangeComplete Members

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChangeComplete Members

The following table shows the **SPasswordChangeComplete** members.

Public Fields

Field	Description
guidTrackingId	A GUID containing the tracking ID from the original PASSWORD_CHANGE.
ullErrorCode	An integer containing an error code.
bstrErrorMessage	A string containing an error message.

See Also

Reference

SPasswordChangeComplete Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChangeComplete Fields

The fields of the **SPasswordChangeComplete** interface is listed in the following table. For a complete list of **SExternal** interface members, see SPasswordChangeComplete Members.

Public Fields

Field	Description
guidTrackingId	A GUID containing the tracking ID from the original PASSWORD_CHANGE.
ullErrorCode	An integer containing an error code.
bstrErrorMessage	A string containing an error message.

See Also

Reference

SPasswordChangeComplete Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SPasswordChangeComplete.guidTrackingId Field

A GUID containing the tracking ID from the original PASSWORD_CHANGE.

Syntax
C++

```
public: GUID guidTrackingId;
```

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChangeComplete Structure (COM)
SPasswordChangeComplete Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChangeComplete.ullErrorCode Field

An integer containing an error code.

Syntax
C++

```
public: ULONGLONG ullErrorCode;
```

Remarks

The string must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChangeComplete Structure (COM)
SPasswordChangeComplete Members
Other Resources
Programming with Enterprise Single Sign-On

SPasswordChangeComplete.bstrErrorMessage Field

A string containing an error message.

Syntax
C++

```
public: BSTR bstrErrorMessage;
```

Remarks

Can be NULL. BSTR must be freed after use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SPasswordChangeComplete Structure (COM)
SPasswordChangeComplete Members
Other Resources
Programming with Enterprise Single Sign-On

SSO_NOTIFICATION_TYPE Enumeration (COM)

Specifies the different notification types used for Enterprise Single Sign-On (SSO).

Syntax

VB

```
<Serializable>
Public Enum SSO_NOTIFICATION_FLAG
```

C#

```
[Serializable]
public enum SSO_NOTIFICATION_FLAG
```

C++

```
[Serializable]
__value public enum SSO_NOTIFICATION_FLAG
```

JScript

```
            public
   Serializable
enum SSO_NOTIFICATION_FLAG
```

Members

Member Name	Value	Direction	Description
SSO_NOTIFICATION_TYPE_NONE	0x00000000	ENTSSO to adapter	No notifications are pending. This notification type is supported by group adapters. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_SHUTDOWN	0x00000001	ENTSSO to adapter	The ENTSSO service requires the adapter to shutdown. The adapter should respond by calling ISSONotification.ShutdownAdapter Method. This notification type is supported by group adapters. It indicates that the group adapter only should shut down. Each individual adapter that is part of the adapter group gets its own SHUTDOWN notification. It is not necessary to confirm this notification.

SSO_NOTIFICATION_TYPE_SHUTDOWN_COMPLETE	0x00000002	ENTSSO to adapter	The ENTSSO service has processed the **ShutdownAdapter** method. This is the last notification that is received by the adapter using the current event handle. This notification type is supported by group adapters. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE	0x00000003	Both	A password has changed. When sending a password change from an adapter to ENTSSO, the password change is considered to be accepted, and durable, on return from the notification to ENTSSO through SendNotification. That does not indicate that the password change is complete, but rather that it has been accepted, is durable, and is likely to be accomplished. Under some error conditions queues could fill up, in which case older password changes could be discarded. In general, you should perform more recent password changes than older ones. An adapter should make password changes received through ReceiveNotification, The adapter should make that change durable or effect the change immediately. The adapter should then call **SendNotification** with PASSWORD_CHANGE_COMPLETE for that password change to delete it from the database queue. This notification type is not supported by group adapters. An adapter must confirm this notification. It is optional for ENTSSO to confirm this notification.
SSO_NOTIFICATION_TYPE_PASSWORD_CHANGE_COMPLETE	0x00000004	Both	The ENTSSO service completed the password change. Note that the definition of complete may vary depending on what is configured. For more information, see the Remarks section of ReceiveNotification. You can use the tracking ID to correlate the original request with the response. This notification type is not supported by group adapters. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_PASSWORD_EXPIRED	0x00000005	Both	The password for an account has expired. For ENTSSO, this causes the stored password for this account to be flagged as **expired**. This notification type is not supported by group adapters. It is necessary for an adapter to confirm this notification. It is optional for ENTSSO to confirm this notification. Note that this notification is currently not implemented. ENTSSO will ignore any SSO_NOTIFICATION_TYPE_PASSWORD_EXPIRE messages, and will never send a message of this type to an adapter.
SSO_NOTIFICATION_TYPE_STATUS_REQUEST	0x00000006	Both	An inquiry regarding the status of the adapter or ENTSSO service, or as a response. If the ENTSSO service detects that a password sync adapter has not sent any password changes for some time, it may send a STATUS notification to that adapter as a "keep alive". The adapter should respond with a STATUS_ONLINE or STATUS_OFFLINE notification. This notification type is supported by group adapters. It applies only for the status of the group adapter, not for the adapters within the adapter group. It is not necessary to confirm this notification.

SSO_NOTIFICATION_TYPE_STATUS_ONLINE	0x00000007	Both	The status of an adapter or ENTSSO service is online. If the ENTSSO service detects that a password sync adapter has not sent any password changes for some time, it may send a STATUS notification to that adapter as a "keep alive". If online, the adapter should respond with a STATUS_ONLINE. If the adapter detects that it is offline, it can send these notifications unsolicited. This notification type is supported by group adapters. It applies only for the status of the group adapter, not for the adapters within the adapter group. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_STATUS_OFFLINE	0x00000008	Both	The adapter or ENTSSO service is offline. If the ENTSSO service detects that a password sync adapter has not sent any password changes for some time, it may send a STATUS notification to that adapter as a "keep alive". If your adapter is offline, it should respond with a STATUS_OFFLINE notification. If the adapter detects that it is offline, it can send these notifications unsolicited. This notification type is supported by group adapters. It applies only for the status of the group adapter, not for the adapters within the adapter group. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_ADAPTERS_IN_GROUP	0x00010000	ENTSSO to adapter	An adapter is contained within a specified adapter group. It is one of the first notifications received by a group adapter after initialization. The adapter names are contained within the "new external credentials array" parameter. If there are a very large number of adapters in this adapter group, it is possible that this notification is received by the group adapter more than once with the remaining adapter names. This notification type is only issued to group adapters. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_ADD_ADAPTER	0x00010001	ENTSSO to adapter	An adapter has been added to the adapter group. The adapter name that has been added is contained within the "notification string" parameter. This notification type is only issued to group adapters. It is not necessary to confirm this notification.
SSO_NOTIFICATION_TYPE_DELETE_ADAPTER	0x00010002	ENTSSO to adapter	An adapter has been deleted from the adapter group. The adapter name that has been deleted is contained within the "notification string" parameter. This notification type is only issued to group adapters. It is not necessary to confirm this notification.

Remarks

There is no online or offline notification for adapters to the group adapter. This is because control of the individual adapters is handled by each adapter itself.

In general, the notifications that require confirmation are those that are durable in the database queue. The other notifications are control and status information which are transient. Notifications that require confirmation have one of the confirm flags set.

There is no notification to or from the adapter about enabled or disabled status. This is because the enable and disable is handled by the enable and disable of the underlying configuration store application. Thus, when the adapter is disabled by an administrator, the adapter receives access denied messages from all calls to ENTSSO.

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Other Resources

Programming with Enterprise Single Sign-On

SSO_NOTIFICATION_FLAG Enumeration (COM)

Specifies the different types of flags used for Enterprise Single Sign-On (SSO).

Syntax

VB

```
<Serializable>
Public Enum SSO_NOTIFICATION_FLAG
```

C#

```
[Serializable]
public enum SSO_NOTIFICATION_FLAG
```

C++

```
[Serializable]
__value public enum SSO_NOTIFICATION_FLAG
```

JScript

```
            public
   Serializable
enum SSO_NOTIFICATION_FLAG
```

Remarks

The following table describes the possible values for the notification flags.

Members

Member Name	Value	Direction	Description
SSO_NOTIFICATION_FLAG_NONE	0x00000000	Both	Null value.
SSO_NOTIFICATION_FLAG_ADMIN_CHANGE	0x00000001	Both	The password change was a result of an administrator action. Some systems will be able to distinguish between an administrator action, while some will not. ENTSSO will not set this flag. This flag is currently not used.
SSO_NOTIFICATION_FLAG_TEST	0x00000008	Both	The current notification is a test request. Your adapter can safely ignore this notification. You can use this notification for testing and diagnostics purposes. This flag is currently not used.
SSO_NOTIFICATION_FLAG_AUDIT	0x00000010	Both	The current request requires auditing. This flag is currently not used.

SSO_NOTIFICATION_FLAG_WINDOWS	0x00000020	Not applicable	Reserved for internal use.
SSO_NOTIFICATION_FLAG_WAIT	0x00000040	Adapter to ENTSSO	**ReceiveNotification** should block and wait until a notification is available. Best practice indicates you should set this flag and allow the PS Helper to wait for notifications.
SSO_NOTIFICATION_FLAG_SEND_ONLY	0x00000080	Adapter to ENTSSO	Indicates that this PS Helper should initialize for sending only. It assumes that another PS Helper will be initialized normally. You should use this flag when using one PS Helper for ReceiveNotification (for receiving password changes), and another fro SendNotification (for sending password changes).

Requirements

Type Library: SSOPSHelper 1.0 Type Library (SSOPSHelper.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Other Resources

Programming with Enterprise Single Sign-On

SStatus Structure (COM)

Describes the current status.

Syntax

```
public: __value struct SStatus
```

Requirements

Type Library: SSOLookup 1.0 Type Library (SSOLookup.dll)

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also

Reference

SStatus Members

Other Resources

Programming with Enterprise Single Sign-On

SStatus Members

The following table shows the **SStatus** members.

Public Fields

Field	Description
ullErrorCode	An integer that contains an error code.
bstrErrorMessage	A string that contains an error message.

See Also

Other Resources

Programming with Enterprise Single Sign-On

SStatus Fields

The fields of the **SStatus** interface are listed in the following table. For a complete list of **SExternal** interface members, see SStatus Members.

Public Fields

Field	Description
ullErrorCode	An integer that contains an error code.
bstrErrorMessage	A string that contains an error message.

See Also

Reference

SStatus Structure (COM)

Other Resources

Programming with Enterprise Single Sign-On

SStatus.ullErrorCode Field

An integer that contains an error code.

Syntax
C++

```
public: ULONGLONG ullErrorCode;
```

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SStatus Structure (COM)
SStatus Members
Other Resources
Programming with Enterprise Single Sign-On

SStatus.bstrErrorMessage Field

A string that contains an error message.

Syntax
C++

```
public: BSTR bstrErrorMessage;
```

Remarks

bstrErrorMessage Can be NULL. You must free **bstrEerrorMessage** after every use.

Requirements

Platforms: Windows XP Professional, Windows Vista, Windows Server 2003, Windows Server 2008

See Also
Reference
SStatus Structure (COM)
SStatus Members
Other Resources
Programming with Enterprise Single Sign-On

Enterprise Single Sign-On Flags

The following flags are used with the Microsoft® Host Integration Server Enterprise Single Sign-On (SSO) methods.

Common flags

Member name	Value
SSO_FLAG_NONE	0x00000000
SSO_FLAG_REFRESH	0x00000001
SSO_FLAG_ENABLED	0x00000002
SSO_FLAG_RUNTIME	0x00000004
SSO_FLAG_SSO_EXTERNAL_TO_WINDOWS	0x00000008
SSO_FLAG_SSO_VERIFY_EXTERNAL_CREDS	0x00000010
SSO_FLAG_ALLOW_TICKETS	0x00000020
SSO_FLAG_VALIDATE_TICKETS	0x00000040
SSO_FLAG_ADMIN_ENABLED	0x80
SSO_FLAG_READ_MODIFY_WRITE	0x100
SSO_FLAG_REPLAY	0x100

Password synchronization flags

Member name	Value
SSO_FLAG_PARTIAL_SYNC_FROM_WINDOWS_TO_DB	0x00000100
SSO_FLAG_PARTIAL_SYNC_FROM_EXTERNAL_TO_DB	0x00000200
SSO_FLAG_FULL_SYNC_FROM_WINDOWS_TO_EXTERNAL	0x00000400
SSO_FLAG_FULL_SYNC_FROM_EXTERNAL_TO_WINDOWS	0x00000800
SSO_FLAG_SYNC_VERIFY_EXTERNAL_CREDS	0x00001000
SSO_FLAG_SYNC_PROVIDE_OLD_EXTERNAL_CREDS	0x00002000
SSO_FLAG_SYNC_ALLOW_MAPPING_CONFLICTS	0x00004000

Application flags

Member name	Value
SSO_FLAG_APP_GROUP	0x10000
SSO_FLAG_APP_USES_GROUP_MAPPING	0x00010000
SSO_FLAG_APP_EXTERNAL_NAME_SAME	0x00020000
SSO_FLAG_APP_ALLOW_LOCAL	0x40000

SSO_FLAG_APP_ADMIN_SAME	0x80000
SSO_FLAG_APP_CONFIG_STORE	0x100000
SSO_FLAG_APP_TICKET_TIMEOUT	0x200000
SSO_FLAG_APP_ADAPTER	0x400000
SSO_FLAG_APP_FILTER_BY_TYPE	0x1
SSO_FLAG_APP_SENSITIVE_INFO_REMOVED	0x80000000
SSO_FLAG_APP_DIRECT_PASSWORD_SYNC	0x2000000
SSO_FLAG_APP_WINDOWS_CREDS	0x800000
SSO_FLAG_APP_RESTRICTED_CREDS	0x1000000

Application Type flags

Member name	Value
SSO_APP_TYPE_NONE	0
SSO_APP_TYPE_INDIVIDUAL	0x1
SSO_APP_TYPE_GROUP	0x2
SSO_APP_TYPE_CONFIG_STORE	0x4
SSO_APP_TYPE_HOST_GROUP	0x8
SSO_APP_TYPE_PS_ADAPTER	0x10
SSO_APP_TYPE_PS_GROUP_ADAPTER	0x20

Mapping flags

Member name	Value
SSO_FLAG_MAPPING_REQUIRES_WINDOWS_PASSWORD	0x01000000
SSO_FLAG_MAPPING_REQUIRES_EXTERNAL_CREDS	0x02000000
SSO_FLAG_MAPPING_ENABLE_AUDIT	0x04000000
SSO_FLAG_MAPPING_CONFIG_STORE	0x8000000
SSO_FLAG_MAPPING_ADMIN	0x10000000
SSO_FLAG_MAPPING_HOSTGROUP	0x20000000
SSO_FLAG_MAPPING_GROUP	0x40000000
SSO_FLAG_MAPPING_HIDE	0x80000000
SSO_FLAG_MAPPING_CHECK	0x100000

Field information flags

Member name	Value
SSO_FLAG_FIELD_INFO_MASK	0x10000000
SSO_FLAG_FIELD_INFO_SYNC	0x20000000

Requirements

Platforms: MicrosoftWindows Server™ 2003, Windows® XP Professional, Windows 2000 Server

Samples

This section of Microsoft Host Integration Server 2009 Help provides information for the samples contained in the Host Integration Server 2009 SDK.

In This Section

Adapter Samples

Application Integration Samples

Data Integration Samples

End-to-End Scenario Sample

Messaging Samples

Network Integration Samples

Single Sign-On Samples

See Also

Other Resources

Development

Adapter Samples

The Adapter Samples section describes the sample applications located in the <directory>\Program Files\Microsoft Host Integration Server 2009\SDK\Samples\Adapter directory.

In This Section

Host Applications Samples

MQSC Adapter Samples

Reference

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Reference

Related Sections

BizTalk Adapters

See Also

Other Resources

Samples

Host Applications Samples

The HostApplications sample demonstrates the BizTalk Adapter for Host Applications.

Location in SDK

<installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\Adapter\HostApplications

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the root directory	Contains detailed setup, configuration, deployment, and execution instructions.
In the \BtsMsiFiles directory	BizTalk Server installation files.
In the \CICSPrograms directory	The cobol files for the sample application.
In the \Documents directory	Contains the XML documents used by BizTalk.
In the \FiledropApplication directory	The Visual Studio files for creating a file drop application.
In the \HTTPApplication directory	The Visual Studio files for creating an http application.
In the \InputDocuments directory	The directory to drop input documents for the sample application.
In the \OutputDocuments directory	The directory for the application to drop documents to.
In the \TIConfiguration directory	Contains components used to configure TI.

See Also

Other Resources

Adapter Samples

MQSC Adapter Samples

This section describes the samples provided for the BizTalk Adapter for WebSphere MQ.

In This Section

Pipeline Component Sample

BizTalk Correlation Sample

Related Sections

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Reference

Client-Based BizTalk Adapter for WebSphere MQ Programmer's Guide

BizTalk Adapter for WebSphere MQ

See Also

Other Resources

Network Integration Samples

Pipeline Component Sample

The Pipeline component sample describes the three components necessary to put messages into an MQSeries queue with the MQRFH2 header: a pipeline component project, a pipeline project that makes use of the pipeline component, and a test application that puts messages into the queue.

Location in the SDK

<Installation directory>\SDK\Samples\Adapters\MQSC\MQRFH2Sample

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the / BizTalk ServerMQRFH 2Pipeline folde r	Describes the BizTalk pipeline project that contains both receive and send pipeline.
In the / MQRF H2PipelineCo mponent folde r	Contains the pipeline component that reads MQRFH2 properties from the MQSeries message and promotes it into BizTalk context properties in the receive scenario. Similarly, on the send side, the component can set these MQRFH2 properties in the MQSeries message based on values provided through the context properties..
In the MQRFH 2TestDriver fol der	Contains an application that can send MQSeries messages to an MQSeries queue with the MQRFH2 header.

How to Use the Sample

Use the following instructions to build, deploy, configure, and run the sample.

To build and deploy the sample

1. Open MQRFH2TestDriver.cpp and specify the correct channelName, connectionName, qmgrName and qName to correct values that match what you have configured in your WebSphere MQ Server configuration.
2. Save the file.
3. Run <InstallPath>\SDK\Samples\Adapters\MQSC\MQRFH2Sample\setup.bat to build the projects and deploy the pipeline component and pipeline project in BizTalk.

To configure and run the sample

1. Create a BizTalk receive port and receive location using MQSC adapter to point to an MQ queue on a MQ Server (MQServer1\QM1\RECVQ).
2. Associate MQRFHReceivePipeline with this receive location.
3. Configure a send port using MQSC Adapter to point to an MQ queue on a MQ Server (MQServer1\QM2\SENDQ).
4. Create a subscription that uses the receive port created in step1.

   This will enable a round trip receive-send scenario between MQ->BizTalk->MQ. Associate MQRFHSendPipeline with this send port.
5. Enable the receive location and start the send port.
6. Start the host instance associated with these end-points.
7. Use MQRFH2TestDriver.exe to put an MQSeries message to MQServer1\QM1\RECVQ.
8. Check the MQSeries queue MQServer1\QM2\SENDQ to see the new message that was sent from BizTalk Server.

   You can view the message in MQSeries queue to see the MQRFH2 header.

See Also

Other Resources

MQSC Adapter Samples

BizTalk Correlation Sample

The BizTalk Correlation sample describes how to retrieve messages from an MQSeries queue and put the response messages into another MQSeries queue.

Location in the SDK

<Installation directory>\SDK\Samples\Adapters\MQSC\ MQSSolicitResponse

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /MQSCorrelationSetOrchestrationWithSolicitResponse folder	The BizTalk Orchestration project that contains associations between file and MQSeries end-points along with the correlation sets.
In the / MQSSolicitResponseApp folder	The application that retrieves messages from an MQSeries queue and puts a response message to another MQSeries queue.

How to Use the Sample

Use the following procedures to build, deploy, configure, and run the sample.

To build and deploy the sample

1. On your BizTalk Server computer, open the orchestration project MQSCorrelationSetOrchestrationWithSolicitResponse\MQSCorrelationSolicitResponse.sln.
2. Open the orchestration MQSCorrelationSolicitResponse.odx.
3. Double-click on the message assignment shape MessageAssignment_1.
4. Adjust the assignment statements for the MQMD_ReplyToQ and MQMD_ReplyToQMgr context properties to point to MQServer1\QM1\QUEUEB.
5. Save the changes to this orchestration.
6. Run MQSCorrelationSetOrchestrationWithSolicitResponse\Setup.bat to build and deploy the orchestration to BizTalk.
7. Run MQSSolicitResponseApp\Setup.bat to build the test application.

To configure and run the sample

1. Create a solicit-response send port, point it at MQServer1\QM1\QUEUEA
2. Create a receive port and a receive location, point it at MQServer1\QM1\QUEUEB
3. Bind the orchestration you deployed earlier to the solicit-response send port and receive port.
4. Create three folders: c:\temp\pickup2, c:\temp\moveit, and c:\temp\dropit2
5. Start the receive locations, send ports and the orchestration.
6. Put an XML file in the pickup2 folder.

   For example, <test>This is a test</test>.
7. Observe the file disappear as it is picked up by BizTalk.
8. Observe the message arrive in MQServer1\QM1\QUEUEA.
9. Observe the MQSeries response message arrive in c:\temp\moveit.

   This message will be an XML file describing the message ID and correlation ID the MQSeries server assigned to the message.
10. Observe that there is one active orchestration instance in the BizTalk Administration Console.

    This is the long-running orchestration waiting for a message to correlate with the message that it already received.

11. Use the test application MQSSolicitResponseApp to read the message BizTalk sent and send a response message.

    You may also use the command line parameters to receive the message from MQServer1\QM1\QUEUEA.

12. Observe a message arrive in c:\temp\dropit2.

    This message is the response message from the test application.

See Also

Other Resources

MQSC Adapter Samples

Application Integration Samples

This section provides information about the samples that use Transaction Integration to integrate applications between the client and the host server.

In This Section

COMTIIntrinsic Sample

Host-Initiated Processing Samples

Windows-Initiated Processing Samples

Related Sections
See Also
Other Resources
Samples

COMTIIntrinsic Sample

The COMTIIntrinsic sample is a single VBScript file that sets the COMTIIntrinsic value for a specified application.

Location in SDK

<drive>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\COMTIIntrinsic.

See Also

Other Resources

Application Integration Samples

Host-Initiated Processing Samples

Transaction Integrator supports host-initiated processing (HIP) that is, a workflow in which a host-based application is a client to a COM-based or .NET Framework-based server program running on the Microsoft Windows operating system. The HostInitiated folder contains samples designed to demonstrate various aspects of HIP.

In This Section

Batch Sample

CICS Sample

OS400 Sample

See Also

Other Resources

Application Integration Samples

Batch Sample

The Batch sample describes how to use TCP/IP HIP client programs in conjunction with batch files on the host server.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIngetration\HostInitiated\Batch

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /BatchBanking folder BatchBanking.csproj BatchBanking.sln BatchBanking.suo BatchBAnkingException.cs Class1.cs Meadme.txt TI_HIP_SQLServer.sql	The files that contain the sample client application. The readme files describe the different setup and support files necessary for the various configurations. The SQL file contains the sample database.
In the /BatchBanking/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.
In the /BatchBanking/TIClientDefs folder alloc.jcl delete.jcl gaccclie.jcl gbalcle.jcl getaccoud.cbl getaccud.ldk getbalk.cbl getbalk.lkd mshiplkb.cbl mshiplkb.lkd mshipudb.cbl mshipudb.lkd prtvsam.jcl TIClientDefs.tiproj	Contains the batch files for the remote host.

In the /BatchBanking /TIServerDefs BatchBanking.DLL BatchBanking.TIM TIServerDefs.tiproj	Contains the support files that describe the remote host application to the sample application.

See Also

Other Resources

Application Integration Samples

CICS Sample

The CICS sample describes how to compile and access the Woodgrove Bank CICS using host-initiated processing.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\HostInitiated\CICS

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /CICSBanking folder CICSBanking.csproj CICSBanking.sln CICSBanking.suo CICSBankingExceptions.cs Class1.cs Readme-01-VTAM-LU0-LU2-Setup.txt Readme-02-CICS-LU0-LU2-Setup.txt Readme-06-Configure-HIS.txt ReadMeSNA.txt REadMeTCPIP.txt	The files that contain the sample application. The readme files describe the different setup and support files necessary for the various configurations.
In the /CICSBanking/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.

In the /CICSBanking/TIClientDefs folder MSHIPLNK.cbl MSHIPLNK.lkd TIClientDefs.tiproj WBCLKNPS.cbl WBCLKNPS.lkd WBCLKNPT.cbl WBCLNKNPT.lkd WBCUDPCS.cbl WBCUBPCS.lkd WBCUDPCT.cbl WBCUDPCT.lkd wgbmaps.bms wgmaps.lkd	Contains the files that describe the programs, transitions, and maps for the host application.
In the /CICSBanking/TIServerDefs CICSBanking.DLL CICSBanking.TIM TIServerDefs.tiproj	Contains the support files that describe the remote host application to the sample application.

OS400 Sample

The OS400 sample describes how to verify an installation of an application on a remote host.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\HostInt\OS400

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /OS400/OS400Banking folder Class1.cs OS400Banking.csproj OS400Banking.sln OS400Banking.suo OS400BankingException.cs ReadMeTCPIP.txt	The files that contain the sample application. The readme files describe the different setup and support files necessary for the various configurations.
In the /OS400Banking/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.
In the /OS400/TIClientDefs folder Mshiplnk.rpg qrpglesrc_ernno_h.txt qrpglesrc_socket_h.txt qrpglesrc_socketutil_h.txt qrpglesrc_socketutilr4.txt TIClientDefs.tiproj wbclknpt.rpg wgbmaps.scr	Contains the files that describe the remote host application.
In the /OS400/TIServerDefs folder OS400Banking.DLL OS400Banking.TIM TIServerDefs.tiproj	Contains the support files that describe the remote host application to the sample application.

See Also

Other Resources

Application Integration Samples

Windows-Initiated Processing Samples

This section describes three sets of samples showing you how to use Windows®-initiated processing.

In This Section

BasicScenarios Sample

Bounded Recordsets Sample

CPlusPlus Sample

DiscriminatedUnions Sample

DotNetRemoting Sample

IMSConnect Sample

InstallationVerification Sample

OS400DPCWithSecurityOverride Sample

PersistentConnections Sample

REOverride Sample

SampleMainframeCode Sample

TIExceptionHandling Sample

Transactions2PC Sample

WebServiceUsingReturnValue Sample

See Also

Other Resources

Application Integration Samples

BasicScenarios Sample

The BasicScenarios code samples demonstrate how to create, set up, configure, and run a complete C++, C#, or VB.NET application using Application Integration technologies.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\BasicScenarios

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the root folder	The source files to build the general TI solution. The readme contains the specific instructions on how to set up and compile the sample.
In the /CPlusPlus Client folder	The files specific to the C++ solution.
In the CSharpClient folder	The files specific to the C# solution.
In the TIHostApplicationDef folder	The support files that define the remote connection and system for the sample. Use these files, along with the SimHost.exe application, to simulate a remote system. For more information, see the readme.
In the VBNetClient folder	The files specific to the VB.NET solution.

See Also

Other Resources

Windows-Initiated Processing Samples

Bounded Recordsets Sample

The Bounded Recordsets sample demonstrates how to use Transaction Integrator (TI) with Microsoft Visual Basic bounded recordsets. This sample includes Visual Basic code and Customer Information Control System (CICS) COBOL code showing how to use bounded recordsets by calling into a CICS transaction program through LU 6.2 (Remote Environment CICS using LU 6.2).

Location in SDK

<Installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\BoundedRecordsets\

File Inventory

The following table shows the files in this sample and describes their purpose.

File(s)	Description
In the \BoundedRecordsets\COBOL-CICS folder RecordSetSample.cbl RecordSetSample.tlb	A TI type library (.tlb file) that can be used with this sample. The type library is set up for accessing a transaction named GETI on the host. This folder also contains sample COBOL code that can be compiled and linked on the mainframe side. The compiled code should be set up to run on the host as a transaction named GETI or the TI type library must be changed to reflect the name of the transaction if it is different.
In the \BoundedRecordsets\VB folder RecordSetSample.bas RecordSetSample.exe RecordSetSample.vbp RecordSetSample.vbw	A Visual Basic class file that illustrates the use of bounded recordsets. Note that additional Visual Basic code would need to be written to use this Visual Basic class file in a project. The code in the class file demonstrates how to create a recordset and populate it with data to send to the mainframe. Note that there is no code that actually displays the data that comes back from the mainframe. You can put in a breakpoint in the Visual Basic code using the debugger and use the immediate window to view the data or insert further code to examine the data that is returned.

See Also

Other Resources

Windows-Initiated Processing Samples

CPlusPlus Sample

The CPlusPlus code sample demonstrates how to create, set up, configure, and run a complete C++ application using Application Integration technologies.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\CPlusPlus

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /CPlusPlus folder CPlusPlus.cpp CPlusPlus.h CPlusPlus.rc CPlusPlus.sln CPlusPlus.suo CPlusPlus.vcproj Readme.txt Resource.h StdAfx.cpp StdAfx.h	The source files to build the application. The readme contains the specific instructions on how to set up and compile the sample.
In the /CPlusPlus/TIHostApplicationDef folder BankingELMLink.tlb BankingSNALink.tlb GetBalance.cbl TIHostAppliationDef.tiproj WIPExportedConfig.xml	The support files that define the remote connection and system for the sample. Use these files, along with the SimHost.exe application, to simulate a remote system. For more information, see the readme.

See Also

Other Resources

Windows-Initiated Processing Samples

DiscriminatedUnions Sample

The Discriminated Unions sample describes how to create an application that uses discriminated unions.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\DiscriminatedUnions

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /DiscriminatedUnions folder DiscriminatedUnions.csproj DiscriminatedUions.sln DiscriminatedUinions.suo Program.cs Readme-01-Setup.txt Readme-02-Step-By-Step.txt	The file necessary to create and build the DiscriminatedUnions sample. Readme-01-Setup.txt contains instructions on how to set up and build the application, while Readme-02-Step-By-Step.txt describes how to recreate the entire sample manually.
In the /DiscriminatedUnions/TIHostApplicationDef folder Banking.DLL GetAInfo.cbl TIHostApplicationDef.tiproj WIPExportedConfig.xml	The support files that define the remote system and data for the sample.

See Also

Other Resources

Windows-Initiated Processing Samples

Host Integration Server Designer Discriminated Union Tutorials

In This Section

Tutorial 1: Creating a Project that Uses Discriminated Unions

This tutorial provides a simple walkthrough on how to build a project that uses discriminated unions. This tutorial also sets up environmental parameters that you can use in the second tutorial.

In This Section

Import the Discriminated Union Tutorial into TI Manager

To follow the steps in Tutorial 1: Creating a Project that Uses Discriminated Unions, first, you must import the files necessary to run the tutorial.

To import the discriminated union tutorial

1. In TI Manager, expand the **Transaction Integrator** node.
2. Right-click the **Window-Initiated Processing** node, and then click **Import**.
3. On the **Welcome** page of the Import WIP Definitions Wizard, click **Next**.
4. On the **Define Import characteristics** page, confirm that the **Use Original Definitions** radio button is selected.
5. Use **Browse** to locate the TIHostApplicationDef folder, and then click **Next**.

   The TIHostApplicationDef folder contains all of the relevant files for describing the host environment for this tutorial. The folder is located in <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\DiscriminatedUnions\TIHostApplicationDef.
6. On the **Importing WIP Definitions** page, wait for the import to complete, and then click **Next**.
7. On the **Completing the Import WIP Definitions Wizard** page, click **Finish**.

See Also

Concepts

Start SimHost for the Discriminated Union Tutorial

Other Resources

Tutorial 1: Creating a Project that Uses Discriminated Unions

Start SimHost for the Discriminated Union Tutorial

The second step in Tutorial 1: Creating a Project that Uses Discriminated Unions turns on SimHost, which simulates a remove mainframe for the tutorial to connect to.

Procedures **To start SimHost for the discriminated union tutorial**

1. Right-click **Start**, and then click **Explore**.
2. Locate the SimHost folder.

   For this tutorial, the SimHost folder is located in <*Installation directory*>\Program Files\Microsoft Host Integration Server\System.
3. Double-click SimHost.exe.

   This starts the Microsoft Transaction Integrator Host Simulator. You can use the Host Simulator to simulate a Host Environment. For this tutorial, you will use it to act as a remote Host operating over a TCP/IP CICS connection.
4. Click **Start TCP**.

See Also

Concepts

Build and Execute the Discriminated Union Tutorial

Other Resources

Tutorial 1: Creating a Project that Uses Discriminated Unions

Build and Execute the Discriminated Union Tutorial

Finally, you can build and execute the discriminated union tutorial sample application. After you examine this application, you can create your own application in Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedures **To build and execute the discriminated union tutorial**

1. In Visual Studio, on the **File** menu, click **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, locate the folder that contains the tutorial solution file.

   For this tutorial, the tutorial solution file is located in <Installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\DiscriminatedUnions.
3. Click DiscriminatedUnions.sln, and then click **Open**.
4. Click **Build**, and then click **Build Solution**.
5. Click **Debug**, and then click **Start Debugging**.

   A console window appears and displays the output of the application.
6. End the debugging session by closing the console window.

See Also

Other Resources

Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application

Tutorial 1: Creating a Project that Uses Discriminated Unions

Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application

In this tutorial, you import a COBOL file, create a Discriminated Value Table to map union members, and write code to access the remote host.

In This Section

See Also

Other Resources

Tutorial 1: Creating a Project that Uses Discriminated Unions

Create a New Project for the Discriminated Union Tutorial

To start Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application, you must create a new project to contain your code.

Procedures **To create a new project for the discriminated union tutorial**

1. In Visual Studio, on the **File** menu, select **New**, and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, select **Host Integration Projects**.
3. In the **Templates** pane, select **Transaction Integrator Project**.
4. In the **Name** field, type **DiscrUnionTutorial**.
5. In the **Location** field, type the location where you want to save the tutorial, and then click **OK**.

   For this tutorial, the location of the project will be <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt.

See Also

Concepts

Create the Transaction Integration Project

Other Resources

Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application

Create the Transaction Integration Project

To use a discriminated union, you must create a Transaction Integration project for your solution.

Follow these steps to create a project for
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedures **To create the Transaction Integration project**

1. In Solution Explorer, right-click DiscrUnionTutorial, point to **Add**, and then click **Add .NET Client Library**.
2. On the **Add New Item** dialog box, confirm that .NET Client Library is selected in the **Templates** pane.
3. In the **Name** field, type **Banking**, and then click **Add**.
4. On the **.NET Client Library** page, click **Next**.
5. On the **Library** page, in the **Interface Name** field, type **Accounts**, and then click **Next**.
6. On the **Remote Environment** page, in the **Programming Model** list, select **ELM Link**, and then click **Next**.
7. Click **Create**.

See Also
Concepts
Import the Host Definition File

Import the Host Definition File

After you create a Transaction Integrator project (in Create the Transaction Integration Project), you can import the host definition file.

Follow these steps to import the host definition file for
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedures **To import the host definition file**

1. On the menu bar in Visual Studio, click **View**, and then click **Properties Window**.
2. On the Banking.DLL tab, right-click the **Banking** node, point to **Import**, and then click **Host Definition**.
3. On the Welcome page of the **Import COBOL** wizard, click **Next**.
4. On the **Import COBOL Source File** page, click **Browse**, and then locate the TIHostApplicationDef folder.

   For this tutorial, the TIHostApplicationDef folder is located at <*Installation Directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\DiscriminatedUnions\TIHostApplicationDef.
5. Click the GetAInfo.cbl file, click **Open**, and then click **Next**.
6. On the **Import Options** page, click **Next**.
7. On the **DFHCOMMAREA** page, select the box next to the **DFHCOMMAREA** node, and then click **Next**.
8. Expand the **DFHCOMMAREA** node.
9. Click the arrows next to the **05 SSN** field, and then click **In**.
10. Click the arrows next to the **05 ACCT-ARRAY OCCURS 2 TIMES** field, and then click **In\Out**.
11. Click **Next**.
12. On the **Data Tables, Structures and Unions** page, click **Next**.
13. On the **Completing the Import COBOL Wizard** page, click **Modify**.

See Also
Concepts
Modify the Discriminant Value Table

Modify the Discriminant Value Table

After you import the host definition file (in Import the Host Definition File), you can modify the logic of the associated discriminant value table.

Follow these steps to modify the discriminant value table for
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedures **To modify the discriminant value table**

1. On the Banking.dll tab, expand the **Banking** .dll node, expand the **Accounts interface** node, expand the **GetAInfo method** node, and then expand the **ACCT_ARRAY parameter** node.
2. Right-click the **UNION1** node, and then click **Properties**.
3. In the **Properties** window, in the **Discriminant** field, click the drop-down button, and then click the ACCT_ARRAY.ACCT_TYPE value.
4. Click the **DVT** field, and then click the **ellipsis (...)** button.
5. In the **Discriminant Value Table** dialog box, click the drop-down button under the **Union Member** field, and then click **Checking**.
6. Double-click the field in the **Condition** column, and type **C**.
7. In the **Union Member** column, click the drop-down button under the **Checking** field, and then click Savings.
8. Double-click the **Condition** field under the **C**, and then type **S**.
9. Click **OK**.

See Also
Concepts
Save and Deploy the GetAInfo Interface

Save and Deploy the GetAInfo Interface

After you modify the discriminant value table (in Modify the Discriminant Value Table), you can save and deploy the interface.

Follow these steps to deploy the GetAInfo interface for
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedures **To save and deploy the GetAInfo interface**

1. In the Banking.dll tab, click the **GetAInfo** node.
2. In the **Properties** window, click the **Include Context Parameters** field, and then click **False** in the list.
3. On the **File** menu, click **Save All**.
4. In the Banking .dll tab, click the **Banking** node.
5. In the **Properties** window, click the **Remote Environment** field, and then select **SimHost using ELM Link**.
6. In the Banking.dll tab, right-click the **Banking** node, and then click **Deploy**.

See Also
Concepts
Create a Visual C# Project for the Discriminated Union Tutorial

Create a Visual C# Project for the Discriminated Union Tutorial

After you deploy the GetAInfo interface (in Save and Deploy the GetAInfo Interface), you can create a C# project that can use that interface to access the remote mainframe.

Follow these steps to create a C# project for
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedure **To crate a Visual C# project for the tutorial**

1. In Visual Studio, on the **File** menu, select **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Project Types** pane, click **Visual C#**.
3. In the **Templates** pane, click **Console Application**, and then click **OK**.

See Also
Concepts
Code the C# Application for the Discriminated Union Tutorial

Code the C# Application for the Discriminated Union Tutorial

After you create the C# project to contain your code (in Create a Visual C# Project for the Discriminated Union Tutorial), you can write code for the C# application that uses the deployed interface to access the remote mainframe.

Follow these steps to write code for the application in
Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application.

Procedure **To code the C# application for the discriminated union tutorial**

1. In Solution Explorer, expand the ConsoleApplication1 node.
2. Right-click **References**, and then click **Add Reference**.
3. In the **Add Reference** dialog box, select the **Browse** tab, and locate the DiscrUnionTutorial folder.

   For this tutorial, the DiscrUnionTutorial folder is located at <*Installation Directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\DiscrUnionTutorial\DiscrUnionTutorial.
4. Click Banking.DLL, and then click **OK**.
5. Add the following code to your Program.cs file:

```
using System;
using System.Collections.Generic;
using System.Text;
using Banking;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Banking.Accounts MyBankObj = new Banking.Accounts();
            Banking.ACCT_ARRAY[] accountInfoArray = new Banking.ACCT_ARRAY[2];
            Banking.SAVINGS MySavInfo = new Banking.SAVINGS();
            Banking.CHECKING MyChkInfo = new Banking.CHECKING();

            accountInfoArray[0].ACCT_NUM = "SAV1234567";
            accountInfoArray[0].ACCT_TYPE = "S";
            accountInfoArray[0].UNION1 = MySavInfo;

            accountInfoArray[1].ACCT_NUM = "CHK4566112";
            accountInfoArray[1].ACCT_TYPE = "C";
            accountInfoArray[1].UNION1 = MyChkInfo;

            MyBankObj.GetAInfo("11223333", ref accountInfoArray);

            foreach (ACCT_ARRAY aa in accountInfoArray)
            {
                switch (aa.ACCT_TYPE)
                {
                    case "C":
                        Banking.CHECKING ChkInfo = (Banking.CHECKING)aa.UNION1;
                        break;
                    case "S":
                        Banking.SAVINGS SavInfo = (Banking.SAVINGS)aa.UNION1;
                        break;
                }
            }
```

```
            }
        }
    }
```

6. On the **File** menu, click **Save All**.
7. Click **Build**, and then click **Build ConsoleApplication1**.
8. Click **Debug**, and then click **Start**.

See Also

Other Resources

Tutorial 2: A Step-by-Step Guide to Creating a Simple Discriminated Union Application

DotNetRemoting Sample

The DotNetRemoting sample describes how to perform remoting using Application Integration and .NET technologies.

Location in SDK

<installation directory>\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\DotNetRemoting

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /DotNetRemoting folder DotNetRemoting.sln DotNetRemoting.suo Readme.txt Service.asmx Web.config	The main files that describe the application.
In the /DotNetRemoting/App_Code folder Service.cs	Defines the web service for the application.
In the /DotNetRemoting/TIHostApplicationDef folder GetBalance.cbl RemBanking.DLL TIHostApplicationDef.tiproj Web.config	The support files that define the remote environment and data for the sample.

See Also

Other Resources

Windows-Initiated Processing Samples

IMSConnect Sample

The IMSConnect sample demonstrates executing a method call to an IMS program using IMS Connect.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\IMSConnect

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the IMSConnect folder IMSConnect.csproj IMSConnect.sln IMSConnect.suo Program.cs Readme.txt	The files that contain the sample application. The Readme.txt file contains instructions on how to set up, compile, and execute the sample.
In the IMSConnect/Properties folder AssemblyInfo.cs	Contains the assembly information for the project.
In the IMSConnect/TIHostApplicationDef folder GetBalance.cbl IMSBanking.DLL TIHostApplicationDef.tiproj WIPExportedConfig.xml	The support files that describe the remote system and data to the sample application.

See Also

Other Resources

Windows-Initiated Processing Samples

InstallationVerification Sample

The InstallationVerification sample describes how to verify an installation of an application on a remote host.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\CSharpInstallationVerification

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /InstallationVerification folder InstallationVerification.csproj InstallationVerification.sln InstallationVerification.suo Program.cs Readme-01-First.txt Readme-02-VTAM-Setup.txt Readme-03-CICS-SNA-Setup.txt Readme-04-IMS-SNA-Setup.txt Readme-05-Configure-HIS.txt Readme-06-TCPIP-Setup.txt Readme-07-CICS-TCPIP-Setup.txt Readme-08-IMS-TCPIP-Setup.txt Readme-09-OS400-TCPIP-Setup.txt	The files that contain the sample application. The readme files describe the different setup and support files necessary for the various configurations.
In the /InstallationVerification/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.

In the /InstallationVerification/TIHostApplicationDef folder CICSSNALink.cbl CICSSNALink.lkd CICSSNAUserData.lkd CICSTCPUserDAta.cbl CICSTCPUserDAta.lkd IMS.cbl IMS.lkd IMSCONV.cbl IMSCONV.lkd ivp.exe IVP_CICS_ELMLink.dll IVP_CICS_SNALink.dll IVP_CICS_SNAUserData.dll IVP_CICS_TRMLink.DLL IVP_CICS_TRMUserData.DLL IVP_CICS_IMSConnect.dll IVP_CICS_SNAUserData.dll IVP_OS400_DCP.dll Mscmtics.cbl Mscmtics.lkd OS400DCP.rpg TiHostApplicationDefltiproj WIPExportedConfig.xml	Contains the support files that describe the remote host application to the sample application.

See Also

Other Resources

Windows-Initiated Processing Samples

OS400DPCWithSecurityOverride Sample

The OS400PDCWithSecurityOverride sample demonstrates how to use the OS400 DPC security override.

Location in SDK

<installation directory>\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\OS400DPCWithSecurityOverride

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /OS400DPCWithSecurityOverride folder OS400DPCWithSecurityOverride.csproj OS400DPCWithSecurityOverride.sln OS400DPCWithSecurityOverridesuo Program.cs Readme-01-Setup.txt Readme-02-Step-By-Step.txt	Contains the code for the sample application. The Readme files describe the general setup, as well as the step-by-step process of setting up and executing the application.
In the /OS400DPCWithSecurityOverride/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.
In the /OS400DPCWithSecurityOverride/TIHostApplicationDef folder GetBalance.rpg SecureBanking.DLL TIHostApplicationDef.tiproj WIPExportedconfig.xml	Contains the support files that describe the remote host system to the sample application.

See Also

Other Resources

Windows-Initiated Processing Samples

PersistentConnections Sample

The PersistentConnections sample demonstrates using the Transaction Integrator Client Context to provide persistent connection Open, Use, and Close calls.

Location in SDK

<installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\PersistentConnections

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the PersistentConnections folder PersistentConnections.csproj PersistentConnections.sln PersistentConnections.suo Program.cs Readme-01-Setup.txt Readme-02-Step-By-Step.txt	The main files that contain the sample. Readme-01-Setup.txt describes how to set up and run the sample, while Readme-02-Step-By-Step.txt describes how to recreate the sample manually.
In the PersistentConnections /Properties folder AssemblyInfo.cs	The assembly information for the sample.
In the PersistentConnections /TIHostApplicationDef folder GetBalance.cbl PCBanking.DLL TIHostApplicationDef.tiproj WIPExportedConfig.xml	The support files that describe the remote host system and data to the sample.

See Also

Other Resources

Windows-Initiated Processing Samples

REOverride Sample

The REOverride sample demonstrates using the Transaction Integrator (TI) Client Context to provide a Remote Environment (RE) Override.

Location in SDK

<installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\REOverride

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the REOverride folder Program.cs Readme-01-Setup.txt Readme-02-Step-By-Step.txt REOverride.csproj REOverride.sln REOverride.suo	The files that contain the sample application. Readme-01-Setup.txt describes how to set up, compile, and run the sample, while Readme-02-Step-By-Step describes how to build the sample manually.
In the REOverride/Properties folder AssemblyInfo.cs	The assembly information for the sample.
In the REOverride/TIHostApplicationDef folder GetBalance.cbl REOBanking.DLL TIHostApplicationDef.tiproj WIPExportedConfig.xml	The support files that describe the remote host system and data for the sample.

See Also

Other Resources

Windows-Initiated Processing Samples

Tutorial: Creating an RE Override

When your application uses a Transaction Integrator (TI) component, you can specify the remote environment (RE) used by the TI run-time environment. This tutorial describes how to create such an RE override, using the REOverride sample found in the Host Integration Server SDK. After you are finished with this tutorial, you will be able to:

- Set up SimHost
- Import a host application definition
- Create and deploy a host application interface
- Create an application that performs an RE override.

Compiling and Running the REOverride Sample

The following procedures describe how to compile and run the REOverride sample application. After you have run the application once, you can perform the tutorial to rebuild the sample in a step-by-step process.

To start the primary instance of SimHost for the REOverride tutorial

1. Right-click **Start**, and then click **Explore**.
2. Locate the **SimHost** folder.

   For this tutorial, the SimHost folder is located in <*Installation directory*>\Program Files\Microsoft Host Integration Server\System.
3. Double-click **SimHost.exe**.

   This starts the Microsoft Transaction Integrator Host Simulator. You can use the Host Simulator to simulate a host environment. For this tutorial, you will use it to act as a remote host operating over a TCP/IP CICS connection.
4. Click **Options**, and then click **Reset to default values**.
5. Click **Start TCP**.

To start the secondary instance of SimHost for the REOverride tutorial

1. Double-click **SimHost.exe**.
2. Click **Options**, and then click **Reset to default values**.
3. In the **TCP/IP CICS ELM** dialog box, in the **Link Port** field, enter **6511**.
4. Make sure there are no duplicate port numbers being used by the Host Simulator.
5. Click **Start TCP**.

To import the host application definitions for the REOverride tutorial

1. In TI Manager, expand the **Transaction Integrator** node.
2. Right-click the **Window-Initiated Processing** node, and then click **Import**.
3. On the **Welcome to the Import WIP Definitions Wizard** page, click **Next**.
4. On the **Define Import Characteristics** page, confirm that the **Use Original Definitions** option button is selected.
5. Use **Browse** to locate the **TIHostApplicationDef** folder, and then click **Next**.

   The TIHostApplicationDef folder contains all of the relevant files for describing the host environment for this tutorial. The folder is located in <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\REOverride\TIHostApplicationDef.
6. On the **Importing WIP Definitions** page, wait for the import to complete, and then click **Next**.

Note that the RE0Banking.Accounts.1 file is registered to the "SimHost ELM Link" host, and not to the "SimHost ELM Link Secondary" host.

7. On the **Completing the Import WIP Definitions Wizard** page, click **Finish**.

To build and execute the REOverride sample

1. In Visual Studio, on the **File** menu, click **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, locate the folder that contains the tutorial solution file.

   For this tutorial, the tutorial solution file is located in <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\AppInt\REOverride.
3. Click **REOverride.sln**, and then click **Open**.
4. Click **Build**, and then click **Build Solution**.
5. Click **Debug**, and then click **Start Debugging**.

   A console window appears and displays the output of the application. You should see one message for the primary host simulator, and one for the secondary host simulator.
6. End the debugging session by closing the console window.

To cause the REOverride sample to fail

1. click **Stop TCP** on the secondary SimHost.
2. Build and execute the REOverride sample again.

   The call to the primary SimHost will succeed, while the call with the RE Override will fail.

A Step-by-Step Guide to the RE Override Tutorial

After you have set up and run the REOverride sample, you can go back and rebuild the solution manually by creating the TI project, adding a .NET client object, importing the host definition file, deploying the .hdf file, and then creating and coding the application.

Step 1: Create a Transaction Integrator Project

To create a new project for the RE Override tutorial

1. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, in the **Project Types** pane, select **Host Integration Projects**.
3. In the **Templates** pane, select **Transaction Integrator Project**.
4. In the **Name** field, type **REOverrideTutorial**.
5. In the **Location** field, type the location where you want to save the tutorial, and then click **OK**.

   For this tutorial, the location of the project will be <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration.

Step 2: Add a .NET Client Object

To add a .NET client object

1. In Solution Explorer, right-click **REOverrideTutorial**, point to **Add**, and then click **Add .NET Client Library**.
2. In the **Add New Item** dialog box, confirm that **.NET Client Library** is selected in the **Templates** pane.
3. In the **Name** field, type **REOBanking**, and then click **Add**.
4. On the **.NET Client Library** page, click **Next**.
5. On the **Library** page, in the **Interface Name** field, type **GetBalance**, and then click **Next**.
6. On the **Remote Environment** page, in the **Programming Model** list, select **ELM Link**, and then click **Next**.
7. Click **Create**.

Step 3: Import the Host Definition File

To import the host definition file

1. On the menu bar in Visual Studio, click **View**, and then click **Properties Window**.
2. On the **REOBanking.dll** tab, right-click the **REOBanking** node, point to **Import**, and then click **Host Definition**.
3. On the **Welcome to the Import COBOL Wizard** page, click **Next**.
4. On the **Import COBOL Source File** page, click **Browse**, and then locate the **TIHostApplicationDef** folder.

   For this tutorial, the TIHostApplicationDef folder is located at <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\REOBanking\TIHostApplicationDef.
5. Click the **GetBalance.cbl** file, click **Open**, and then click **Next**.
6. On the **Import Options** page, click **Next**.
7. On the **01 DFHCOMMAREA** page, select the check box next to the **DFHCOMMAREA** node, and then click **Next**.
8. Expand the **DFHCOMMAREA** node.
9. Click the arrows next to the **name** field, and then click **In**.
10. Click the arrows next to the **ACCNUM** field, and then click **In**.
11. Click the arrows next to the **ACCBAL** field, and then click **Out**.
12. Click **Next**.
13. On the **Data Tables, Structures and Unions** page, click **Next**.
14. On the **Completing the Import COBOL Wizard** page, click **Modify**.

Step 4: Save and Deploy the Interface

To save and deploy the GetAInfo interface

1. On the **REOBanking.dll** tab, click the **GetBalance** node.
2. In the **Properties** window, click the **Include Context Parameters** field, and then click **True** in the list.
3. On the **File** menu, click **Save All**.
4. On the **REOBanking.dll** tab, click the **REOBanking** node.
5. In the **Properties** window, click the **Remote Environment** field, and then select **SimHost using ELM Link**.
6. On the **REOBanking.dll** tab, right-click the **REOBanking** node, and then click **Deploy**.

Step 5: Create a Visual Studio Project

To create a Visual C# project for the tutorial

1. In Visual Studio, on the **File** menu, point to **Add**, and then click **New Project**.
2. In the **Add New Project** dialog box, in the **Project Types** pane, click **Visual C#**.
3. In the **Templates** pane, click **Console Application**, and then click **OK**.

Step 6: Code the Client Application

To code the C# application for the RE Override tutorial

1. In Solution Explorer, expand the **ConsoleApplication1** node.
2. Right-click **References**, and then click **Add Reference**.
3. In the **Add Reference** dialog box, click the **Browse** tab, and locate the **REOverride** folder.

   For this tutorial, the REOverride folder is located at <*Installation directory*>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\REOverride.
4. Click **REOBanking.dll**, and then click **OK**.
5. Add the following code to your Program.cs file:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.HostIntegration.TI;
using REOBanking;

namespace REOverride
{
    class Program
    {
        static void Main(string[] args)
        {
            object[] contextArray = null;
            decimal Balance = 0.0m;
            ClientContext TIClientContext = new ClientContext();
            REOBanking.Accounts MyBankObj = new REOBanking.Accounts();
            try
            {
                MyBankObj.GetBalance("Kim Akers", "123456", out Balance, ref contextAr
ray);
                Console.WriteLine("Account balance from the primary RE {0,9:C2}\n", Ba
lance);
                TIClientContext.WriteContext("REOverride", "SimHost ELM Link Secondary
", ref contextArray);
                MyBankObj.GetBalance("Kim Akers", "123456", out Balance, ref contextAr
ray);
                Console.WriteLine("Account balance from the Secondary RE {0,9:C2}\n",
Balance);
            }

            catch (Microsoft.HostIntegration.TI.CustomTIException Ex)
            {
                Console.WriteLine("Exception: TI Runtime Error {0}", Ex.Message);
            }
            catch (Exception Ex)
            {
                Console.WriteLine("Exception: {0}", Ex.Message);
                if (Ex.Message.StartsWith("ClassFactory cannot supply requested class"
, StringComparison.CurrentCultureIgnoreCase))
                {
                    Console.WriteLine("Error: REOBanking object could not be created.
Use TI Manager to ensure it is registered");
                }
            }

            Console.WriteLine("\nPress any key to continue...");
            Console.Read();
        }
    }
```

6. }On the **File** menu, click **Save All**.
7. Click **Build**, and then click **Build ConsoleApplication1**.
8. Click **Debug**, and then click **Start**.

SampleMainframeCode Sample

The SampleMainframeCode sample contains sample code for a variety of mainframe environments.

Location in SDK

<installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\SampleMainframeCode

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the SampleMainframeCode folder SampleMainframePrograms.sln SampleMainframePrograms.suo	Visual Studio solution containing the SampleMainframeCode files.
In the SampleMainframeCode/SNA CICS folder	The mainframe files for an SNA CICS sample application.
In the SampleMainframeCode/SNA CICS Link folder	The mainframe files for an SNA CICS Link sample application.
In the SampleMainframeCode/SNA IMS folder	The mainframe files for an SNA IMS sample application.
In the SampleMainframeCode/TCP CICS Concurrent folder	The mainframe files for a TCP CICS Concurrent sample application.
In the SampleMainframeCode/TCP CICS MSLink folder	The mainframe files for a TCP CICS MSLink sample application.
In the SampleMainframeCode/TCP IMS Connect folder	The mainframe files for a TCP IMS Connect sample application.
In the SampleMainframeCode/TCP IMS Explicit folder	The mainframe files for a TCP IMS Explicit sample application.
In the SampleMainframeCode/TCP IMS Implicit folder	The mainframe files for a TCP IMS Implicit sample application.
In the SampleMainframeCode/TCP OS400 DPC folder	The mainframe files for a TCP OS400 sample application.

See Also

Other Resources

Windows-Initiated Processing Samples

TIExceptionHandling Sample

The TIExceptionHandling sample demonstrates the use of the meta data error block.

Location in SDK

<installation directory>\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\TIExceptionHandling

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /TIExceptionHandling folder Program.cs Readme.txt TIExceptionHandling.csproj TIExceptionHandling.sln TIExceptionHandlingsuo	Contains the code for the sample application. The readme.txt file describes how to set up, build, and execute the sample.
In the /TIExceptionHandling/Properties folder AssemblyInfo.cs	Describes the sample assembly information.
In the /TIExceptionHandling/TIHostApplicationDef folder MDEBBanking.DLL mdebgal.cbl mscmtics.cbl TIHostApplicationDef.tiproj WIPExportedConfig.xml	The support files that describe the remote host system for the sample application.

Transactions2PC Sample

The Transactions2PC sample demonstrates the user of two-phase commit (2PC) transactional processing using .NET transactions.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\Transactions2PC

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /Transactions2PC folder Program.cs Readme-03-VTAM-LU62-Setup.txt Readme-04-CICS-LU62-Setup.txt Readme-05-Configure-HIS.txt Readme-06-TI-Setup.txt Readme-07-Step-By-Step.txt Transaction2PC.csproj Transactions2PC.sln Transactions2PC.suo	The source code for the samples. The Setup.txt files describe how to set up, configure, and execute the application based on different hosting environments.
In the /Transactions2PC/Properties folder AssemblyInfo.cs	Contains the assembly information description for the code sample.
In the /Transactions2PC/TIHostApplicationDef folder GetBAl62.cbl GetBalance.cbl TIHostApplicationDef.tiproj Tx2PCBankingLink.DLL Tx2PCBankingUserData.DLL WIPExportedConfig	The support files that describe the remote host system for the sample application.

WebServiceUsingReturnValue Sample

The WebServiceUsingReturnValue sample demonstrates creating and interacting with a Web service by using Application Integration technnologies.

Location in SDK

<installation directory>\Program Files\Microsoft Host Integration Server\SDK\Samples\ApplicationIntegration\WindowsInitiated\WebServiceUsingReturnValue

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /WebServiceUsingReturnValue folder Program.cs Readme-01-Setup-IIS.txt WEbServiceUsingReturnValue.csproj WEbServiceUsingReturnValue.sln WEbServiceUsingReturnValue.suo	The files that contain the code sample. Readme-01-Setup-IIS contains detailed instructions for setting up, configuring, and running the sample.
In the /WebServiceUsingReturnValue/Properties folder AssemblyInfo.cs	Contains the assembly information for the sample application.
In the /WebServiceUsingReturnValue/TIHostApplicationDef folder GetBalance.cbl TIHostApplicationDef.tiproj WSBanking.DLL	The support files that describe the remote host environment for the sample application.

See Also

Other Resources

Windows-Initiated Processing Samples

Data Integration Samples

This section of the Host Integration Server 2009 Developer's Guide describes the sample applications that implement data integration using DB providers, drivers, and Microsoft ActiveX controls.

In This Section

Data Access Samples

Data Queues Sample

File Transfer Sample

Reference

Data Integration Programmer's Reference

Related Sections

Data Integration Programmer's Guide

See Also

Other Resources

Samples

Data Access Samples

The following topics describe the code samples available for the Managed Provider for DB2.

In This Section

ManagedDb2Client Sample

MsDb2WebApp Sample

MsDb2WebService Sample

Reference

Microsoft.HostIntegration.MsDb2Client

Related Sections

Managed Provider for DB2 Programmer's Guide

See Also

Other Resources

Data Integration Samples

ManagedDb2Client Sample

The ManagedDb2Client sample describes how to create a client application that accesses a remote DB2 database.

Location in SDK

<installation directory>\Program Files\<version>\SDK\Samples\DataIntegration\DataAccess\ManagedDb2Client

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /ManagedDb2Client directory AboutBox.cs AboutBox.resx AssemblyInfo.cs MainForm.cs Mainform.resx ManagedDb2client.csproj ManagedDb2client.sln SQLstatement.cs SQLStatement.resx	Contains the primary and support files for creating the client application.

See Also

Reference

Microsoft.HostIntegration.MsDb2Client

Other Resources

Data Access Samples

Managed Provider for DB2 Programmer's Guide

MsDb2WebApp Sample

The MsDb2WebApplication sample describes how to create a web application that accesses a remote DB2 database.

Location in SDK

<installation directory>\SDK\Samples\DataIntegration\DataAccess\MsDb2WebApp

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /MsDb2WebApp directory BankClient.aspx BankClient.aspx.cs BankClient.aspx.resx Global.asax Globabl.asax.resx MsDb2WebApp.sln Web.config	Contains the primary and support files for creating a web application.
In the /MsDb2WebApp/App_Code directory AssemblyInfo.cs Global.asax.cs	Contains the application code files for the web application.
In the /MsDb2WebApp/App_WebReferences/MsDb2WebApp/MsDb2WebService directory BankService.disco BankService.discomap BankService.wsdl	Descriptions of a web service used by the application

See Also

Reference

Microsoft.HostIntegration.MsDb2Client

Other Resources

Data Access Samples

Managed Provider for DB2 Programmer's Guide

MsDb2WebService Sample

The MsDb2WebService sample describes how to create a web service that accesses a remote DB2 database.

Location in SDK

<installation directory>\Program Files\<version>\SDK\Samples\DataIntegration\DataAccess\MsDb2WebService

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /MsDb2WebService directory BankService.asmx BankService.asmx.resx Global.asax Global.asax.resx Key.snk Web.Config	Contains the primary and support files for creating a web service.
In the /MsDb2WebService/App_Code directory AssemblyInfo.cs BankService.asmx.cs Global.asax.cs	Contains the application code files for the web service.

See Also

Reference

Microsoft.HostIntegration.MsDb2Client

Other Resources

Data Access Samples

Managed Provider for DB2 Programmer's Guide

Data Queues Sample

The DataQueue sample uses the AS/400 Data Queue ActiveX control to connect, send, and receive messages to and from an AS/400.

Location in SDK

<installation directory>\Program Files\<version>\SDK\Samples\DataIntegration\DataQueues

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /DataQueues directory AboutBox.cs AboutBox.Designer.cs AboutBox.resx DataQueues.csproj MainForm.cs MainForm.Designer.cs MainForm.resx Program.cs README.txt	Contains the primary and support files for creating the C# application. The readme also contains detailed file descriptions as well as setup, configuration, building, and execution details.
In the /DataQueues/AS_400 directory Dqdemo.chat Dqdemo.dqcreate Dqdemo.dqdelete Dqdemo.dqreadfifo Dqdemo.dqtalk Dqdemo.dqtalklcl	Contains the source files for the target AS/400.
In the /DataQueues/bin/Debug directory DataQueues.vshost.exe	Contains the DataQueues ActiveX control
In the /DataQueues/obje/Debug directory DataQueues.csproj.ResolveComReference.cache Interop.DATAQUEUELib.dll	Support files for the application.

See Also

Other Resources

Data Integration Samples

Managed Provider for DB2 Programmer's Guide

File Transfer Sample

The FileTransfer sample uses the Host File Transfer ActiveX control to connect to and send files to a host system.

Location in SDK

<installation directory>\Program Files\<version>\SDK\Samples\DataIntegration\FileTransfer

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /FileTransfer directory AboutBox.cs AboutBox.Designer.cs AboutBox.resx FileTransfer.csproj MainForm.cs MainForm.Designer.cs MainForm.resx Program.cs Readme.txt	Contains the primary and support files for creating the C# application. The readme also contains detailed file descriptions as well as setup, configuration, building, and execution details.
In the /FileTransfer/bin/Debug directory FileTransfer.vshost.exe	Contains the Host File Transfer ActiveX control
In the /FileTransfer/obj/Debug directory FileTransfer.csproj.ResolveComReference.cache Interop.MSEIGFTLib.dll	Support files for the application.

See Also

Other Resources

Data Integration Samples

Managed Provider for DB2 Programmer's Guide

End-to-End Scenario Sample

The End-to-End Scenario sample describes a complete solution using different Host Integration Server technologies.

Location in the SDK

<installation directory>\Program Files\<version>\SDK\Samples\EndToEndScenarios\WoodgroveBank

File Inventory

File(s)	Description
In the \3270Application folder	A 3270 application for CICS
In the \CustomerCare folder	Contains C# and TI objects for CICS and OS400 that demonstrate a 3270 Application using Transaction Integrator technology. Also contains a C# application using Session Integrator technology to provide screen scraping services against the 3270 application.
In the \Account Management folder	Contains C# and TI objects for Host Files that demonstrate how to populate a VSAM dataset using the Managed Provider for Host Files.
In the \ATM folder	Describes a C# application using Session Integrator technology to provide LUA interaction (ATMs) to CICS.
In the \MainFrameJobs folder	Contains the data for the associated mainframe applications.

See Also

Other Resources

Samples

Messaging Samples

This section describes program samples that demonstrate the use of the extensions and components that make up Microsoft® MSMQ-MQSeries Bridge.

In This Section

- Sample Programs for MSMQ-MQSeries Bridge

Reference

Messaging Programmer's Reference

Related Sections

Messaging Programmer's Guide

See Also

Other Resources

Samples

Sample Programs for MSMQ-MQSeries Bridge

The source code for several sample programs that illustrate using MSMQ-MQSeries Bridge are included in the Host Integration Server 2009 SDK. These files are copied to your hard drive during Host Integration Server software or Host Integration Client software installation when the Host Integration Server Software Development Kit option is selected. These samples are installed in the Samples\AppInt\Bridge subdirectory below where the Host Integration Server SDK software is installed (C:\Program Files\Microsoft Host Integration Server\SDK, by default).

These sample programs include the files in the subdirectories listed in the following table.

Subdirectory	Description
EPRecv	EPRecv Sample A sample program in C that uses the MSMQ-MQSeries Bridge Extensions API to display the MQMD structure in the Message Queuing (also known as MSMQ) extension property.
EPSend	EPSend Sample A sample program in C that uses the MSMQ-MQSeries Bridge Extensions API to override the default MSMQ-MQseries Bridge message property mapping MsgType, ReplyToQMgr, and ReplyToQ in the MQSeries MQMD structure.
MQSRRecv	MQSRRecv Sample A sample program in C that uses the MQSeries API to receive messages from a specified MQSeries queue.
MQSRSend	MQSRSend Sample A sample program in C that uses the MQSeries API to send ten test messages to a specified MQSeries queue.
MSMQRecv	MSMQRecv Sample A sample program in C that uses the Message Queuing API to receive messages from a specified Message Queuing queue.
MSMQSend	MSMQSend Sample A sample program in C that uses the Message Queuing API to sends ten test messages to a specified Message Queuing local or foreign queue.
WMI	WMI MSMQ-MQSeries Bridge Sample A collection of Windows Management Instrumentation (WMI) sample scripts written in Active Server Pages (ASP) that shows how to use WMI to configure the MSMQ-MQSeries Bridge.

Several sample programs with source code are provided with Host Integration Server 2009 that illustrate how to use the MSMQ-MQSeries Bridge and Bridge Extensions.

The MSMQ-MQSeries Bridge samples are designed to be built using Microsoft Visual Studio .NET 2003 or later. Most of these samples also require that the IBM MQSeries Client toolkit be installed, providing access to several MQSeries include and library files.

To build the MSMQ-MQSeries Bridge samples using the command-line compiler, set up your build environment as follows:

- Run VSVARS32.bat from the Visual Studio bin directory (by default, C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools.
- Set the MQS_INC environment variable so it points to the INCLUDE directory where MQSeries was installed. The default location for this variable is normally C:\Program Files\MQSeries Client\tools\c\include.
- Set the MQS_LIB environment variable so it points to the LIB directory where MQSeries was installed. The default location for this variable is normally C:\Program Files\MQSeries Client\tools\lib.

For example, set the following environment variables for building the MQS samples:

```
set MQS_INC=C:\Program Files\MQSeries Client\tools\c\include
set MQS_LIB=C:\Program Files\MQSeries Client\tools\lib
```

To build all the C/C++ samples (EPSend, EPRecv, MQSRRecv, MQSRSend, MSMQSend and MSMQRecv), open a Command Prompt window, navigate to the Bridge subdirectory, and invoke NMAKE. This recursively invokes NMAKE and builds all of the Bridge samples.

To build a specific sample (EPSend, for example), using the command-line compiler, open a Command Prompt window, navigate to the appropriate subdirectory (Bridge\EPSend, for example), and invoke NMAKE.

Note

To build a specific sample (EPSend, for example) using Visual Studio .NET 2003, open the appropriate Visual C++ project file (epsend.vcproj, for example) from the **File** menu. Select a configuration and build the sample from the **Build** menu. Each pro ject file has two configurations, one for a DEBUG build and one for a RETAIL build.

Note

Several of the MSMQ-MQSeries Bridge samples require access to the IBM MQSeries Client toolkit and library files.

The project files for these samples assume that the IBM MQSeries Client toolkit is installed in the default location at C:\Program Files\MQSeries Client\tools. You need to modify the project files if the IBM MQSeries Client toolkit is installed in a different location. For each C source file, you need to change the **Additional Include Directories** property under **C/C++/General**. For the target, you need to change the **Additional Dependencies** property under **Linker/Input**.

In This Section

EPRecv Sample

EPSend Sample

MQSRRecv Sample

MQSRSend Sample

MSMQRecv Sample

MSMQSend Sample

WMI MSMQ-MQSeries Bridge Sample

EPRecv Sample

The MSMQ-MQSeries Bridge Extension API can be used to obtain the original MQSeries message properties for an MQSeries message sent using the MSMQ-MQSeries Bridge to Message Queuing (also known as MSMQ). The Bridge\EPRecv folder contains a sample program written in C that receives messages from a Message Queuing queue using the Message Queuing APIs and prints the original MQSeries message descriptor (MQMD) properties in the PROPID_M_EXTENSION, if they exist. The sample illustrates how to use MSMQ-MQSeries Bridge Extensions. You can use it for testing or troubleshooting the MSMQ-MQSeries Bridge and Bridge Extensions.

The usage for this command-line tool is as follows:

```
EPRecv <computer name>\<queue name>
```

The parameter *<computer name>\<queue name>* is the path name where messages are received. This is a Message Queuing queue name and is specified in UNC or DNS format.

Sample program usage and sample output are as follows:

```
eprecv MSBRIDGE\QUEUE
```

Queue opened.

Waiting for messages to arrive.

Use CTLR-C to stop.

-------> Message arrived:

Label = ''

Body (256) = 'Test Message 0 - 19:41:26'

Body Type (4113)

Extension property found. Dumping values:

MQMD1 Extension Field found. Dumping values:

MQMD1.Report = 00000000 MQMD1.MsgType = 00000008

MQMD1.Feedback = 00000000 MQMD1.Priority = 0

MQMD1.Version = 00000001 MQMD1.Expiry = -1

MQMD1.ReplyToQMgr = 'BRIDGE2K_QM '

MQMD1.ReplyToQ = ' '

MQMD1.UserIdentifier = 'testuser '

MQMD1.ApplIdentityData = ' '

MQMD1.PutApplName = 'n Server\sys'

MQMD1.PutDate = '20000628'

MQMD1.PutTime = '02530720'

MQMD1.MsgId = '414D512053544152 5741525F514D2020 9197523913300000'

MQMD1.CorrelId = '0000000000000000 0000000000000000 0000000000000000'

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

EPSend Sample

You can use the MSMQ-MQSeries Bridge Extension API to override the default MSMQ-MQSeries Bridge message property mapping.

The Bridge\EPSend folder contains a sample written in C that illustrates how to use the MSMQ-MQSeries Bridge Extension API to override the default MSMQ-MQSeries Bridge message property mapping for MsgType, ReplyToQMgr, and ReplyToQ in the MQSeries MQMD structure. This sample sends messages to the Message Queuing (also known as MSMQ) queue, overriding the default values for these MQMD extension fields.

This sample can be used for testing or troubleshooting the MSMQ-MQSeries Bridge and Bridge Extensions.

The usage for this command-line tool is as follows:

```
EPSend <computer name>\<queue name>
```

The parameter <*computer name*>\<*queue name*> is the path name of the specified Message Queuing queue name where messages are to be sent. This Message Queuing queue name can be specified in UNC or DNS format.

Sample program usage and sample output are as follows:

```
epsend IBMNT_QM\QUEUE
```

Queue opened.

Reply Q Manager Name : MSBRIDGE1

Reply Q Name : QUEUE

------> Sending message (Use CTRL-C to stop).Label: ABC

Body: ABC

MQSRRecv Sample

The Bridge\MQSRRecv folder contains a sample program written in C that uses the MQSeries API to receive messages from a specified MQSeries queue. This sample can be used to receive messages sent from MQSeries or sent from Message Queuing (also known as MSMQ) using the MSMQ-MQSeries Bridge. The sample can be used for testing or troubleshooting the MSMQ-MQSeries Bridge.

The usage for this command-line tool is as follows:

```
MQSRRecv <QM name> <queue name>
```

The first parameter, *QM name*, is the name of the MQSeries Queue Manager. The second parameter, *queue name*, is the queue name from which to receive the messages. Note that the program assumes that queue name is located on the *QM name* computer.

You can run the MQSRRecv program on a computer where the MQSeries Client is installed and configured. The environment variables used by the MQSeries client should point to the appropriate channel table file. The computer running the MSMQ-MQSeries Bridge is a good choice because it should already be properly configured.

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

MQSRSend Sample

The Bridge\MQSRSend folder contains a sample written in C that uses the MQSeries API to send ten test messages to a specified MQSeries queue. This sample can be used to send messages to a specified MQSeries queue or to a specified Message Queuing (also known as MSMQ) queue using the MSMQ-MQSeries Bridge. The sample can be used for testing or troubleshooting the MSMQ-MQSeries Bridge.

The usage for this command-line tool is as follows:

```
MQSRSend <local QM name> <destination QM name>
        <queue name>
```

The first parameter, *local QM name*, is the name of the immediate MQSeries Queue Manager to connect to (the server side of the MQI channel). **MQSRSend** needs this information to establish the MQI channel connection.

The second parameter, *destination QM name*, is the destination queue manager for the messages. To send messages to Message Queuing, specify the queue manager alias for the destination QM name representing the Message Queuing queue.

The third parameter, *queue name*, is the name of the queue where the messages should be sent.

You can send MQSeries message to a Message Queuing queue with one of the following methods.

1. Specify the MSMQ-MQSeries Bridge computer name in the destination QM name and the Message Queuing format name in the queue name.

2. Define QREMOTE for the Message Queuing destination QM name in MQSeries.

You can run the MQSRSend program on a computer where the MQSeries Client is installed and configured. The environment variables used by the MQSeries client should point to the appropriate channel table file. The computer running the MSMQ-MQSeries Bridge is a good choice because it should already be properly configured.

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

MSMQRecv Sample

The Bridge\MSMQRecv folder contains a sample written in C that uses the Message Queuing (also known as MSMQ) API to receive messages from a specified Message Queuing queue. This sample can be used to receive messages sent from Message Queuing or receive messages sent from MQSeries using the MSMQ-MQSeries Bridge. The sample illustrates how to receive messages using Message Queuing and can be used for testing or troubleshooting the MSMQ-MQSeries Bridge.

The usage for this command-line tool is as follows:

```
MSMQRecv <computer name>\<queue name>
```

The parameter <*computer name*>\<*queue name*> is the path name of the specified Message Queuing queue name where messages are received. This Message Queuing queue name can be specified in UNC or DNS format.

You can run the MSMQRecv program on any computer where Message Queuing is installed, not necessarily the computer running the MSMQ-MQSeries Bridge.

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

MSMQSend Sample

The SDK\Samples\Bridge\MSMQSend folder contains a sample written in C that sends ten test messages using the Message Queuing (also known as MSMQ) APIs. This sample can be used to send messages to a specified Message Queuing queue or a foreign MQSeries queue through the MSMQ-MQSeries Bridge. The sample illustrates how to send messages using Message Queuing and can be used for testing or troubleshooting the MSMQ-MQSeries Bridge.

The usage for this command-line tool is as follows:

```
MSMQSend <computer name>\<queue name>
```

The parameter <*computer name*>\<*queue name*> is the path name of the specified Message Queuing queue name where messages are to be sent. This Message Queuing queue name can be specified in UNC or DNS format.

You can run the MSMQSend program on any computer where Message Queuing is installed, not necessarily the computer running the MSMQ-MQSeries Bridge.

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

WMI MSMQ-MQSeries Bridge Sample

The Bridge\WMI folder contains a collection of Active Server Pages (ASP) for use with a Web server application that enables you to view and make changes to the MSMQ-MQSeries Bridge configuration using Windows Management Instrumentation (WMI). These sample applications require Microsoft Internet Information Services (IIS) version 3.0 or greater with ASP installed. Host Integration Server 2009 and IIS must be installed and be running on the same computer.

The WMI ASP samples must be installed into the Web server's public directories below WWWRoot. Copy the contents of the entire WMI directory from the SDK\Samples\Bridge\WMI subdirectory to your WWWROOT directory on the Web server. After these files have been copied you should have a WWWROOT\WMI folder containing a number of ASP and GIF files.

The samples may then be run by opening Microsoft Internet Explorer or some other Web browser on the same computer or a different computer and entering the following URL in the address line:

```
http://<computer name>/WMI/WMI_Test_Main.asp
```

Substitute the network name of the computer hosting the Web server and the MSMQ-MQSeries Bridge for the computer name (in angle brackets in the URL above). This will open the main page of the Bridge WMI ASP application and enable you to select any of the other sample ASP pages. Information about each sample is provided on this Web page.

These ASP pages illustrate using WMI to view and make changes to the MSMQ-MQSeries Bridge configuration. The management functions supported by this application enable you to create a new instance, move to other instances (previous and next), delete an instance, and save an instance.

The WMI subdirectory below WWWROOT needs to have IIS security enabled (no anonymous access). Otherwise, the scripts in these subdirectories will fail since the anonymous user account by default does not have access rights that would allow it to start or stop services on Windows 2000 or make changes to the MSMQ-MQSeries Bridge on the Host Integration Server system.

It is possible to host these ASP pages on a computer running the Web server that is different from the computer running the MSMQ-MQSeries Bridge and Host Integration Server. However, this requires some changes to the ASP pages to handle connections to a different computer, security, and authentication issues.

See Also

Other Resources

Sample Programs for MSMQ-MQSeries Bridge

MSMQ-MQSeries Bridge Programmer's Guide

Network Integration Samples

This section of the Host Integration Server 2009 Developer's Guide describes the sample applications that implement APPC, CPI-C, LUA, and SNA print server data filter network integration.

In This Section

Administration and Management Samples

APPC Samples

CPI-C Samples

LUA Samples

SNA Print Server Data Filter Samples

Session Integrator Samples

Reference

Network Integration Programmer's Reference

Microsoft.HostIntegration.SNA.Session

Related Sections

Network Integration Programmer's Guide

See Also

Other Resources

Samples

Administration and Management Samples

The source code for several sample programs that illustrate using Windows Management Instrumentation (WMI) for administration and management of Host Integration Server 2009 are included in the Host Integration Server 2009 SDK. These files are copied to your hard drive during Host Integration Server software or Host Integration Server Client software installation when you select the **Host Integration Server Software Development Kit** option. These samples are installed in the Samples\NetworkIntegration\Administration subdirectory below where the Host Integration Server 2009 SDK is installed (C:\Program Files\Microsoft Host Integration Server\SDK, by default).

These sample programs include the files in the following subdirectories:

File or subdirectory	Description
\VBScript	A WMI sample script written in Microsoft Visual Basic Scripting Edition (VBScript) that illustrates how to import and export configuration information from Host Integration Server.
\Scripts	The main page of a WMI sample script written in Microsoft Active Server Pages (ASP) for retrieving configuration information from Host Integration Server using WMI.
\ASP-SNAWebAdmin	Subsidiary pages of WMI sample scripts written in Microsoft Active Server Pages (ASP) for retrieving configuration information from HIS using WMI. Each one of these subdirectories contains ASP sample scripts that illustrate how to retrieve information from Host Integration Server on a specific feature.

Several sample programs (with source code) that illustrate administration and management are provided with HIS.

In This Section

Active Server Pages SNAWebAdmin Sample

VBScript ImportExport Sample

Reference

Administration and Management Programmer's Reference

Related Sections

Administration and Management Programmer's Guide

See Also

Other Resources

Network Integration Samples

Active Server Pages SNAWebAdmin Sample

The Administration\ASP-SnaWebAdmin folder contains a collection of Active Server Pages (ASP) for use with a Web server application designed to access configuration, management, and status information from the SNA server component of Host Integration Server 2009. These sample applications require Microsoft Internet Information Services (IIS) version 5.0 or later with ASP installed. Host Integration Server 2009 and IIS must be installed and running on the same computer.

The Windows Management Instrumentation (WMI) ASP samples must be installed into the public directories of the Web server below WWWRoot. Copy the contents of the Admin directory from the SDK\Samples\NetworkIntegration\Administration subdirectory, including SNAWebAdmin subdirectory, to your WWWROOT directory on the Web server. After these files have been copied, you should have a copy of the SNAWMI.ASP, fphover.class, and fphoverx.class files in WWWROOT and a WWWROOT\SNAWebAdmin folder with a series of subdirectories containing several ASP and Graphics Interchange Format (GIF) files.

You can then run the samples by using Microsoft Internet Explorer, or some other Web browser on the same computer, or a different computer. Type **HTTP://<computer name>/SNAWMI.asp** in the **Address** box.

Substitute the network name of the computer hosting the Web server and HIS for the computer name in angle brackets in the URL above. This opens the main page of the SNAWebAdmin ASP application and enables you to select any of the other sample ASP pages. Information about each sample is provided on this Web page. Additional information is available at HTTP://<*computer name*>/admin/headers/welcome.htm.

These ASP pages illustrate using WMI to retrieve SNA management and configuration from Host Integration Server 2009.

The two Java class files, fphover.class and fphoverx.class, are redistributable files that are included with Microsoft FrontPage. These files are used in some of the WMI sample scripts instead of a **Submit** button to stop and start services.

Several subdirectories below SNAWebAdmin must have IIS security enabled (no anonymous access); otherwise the scripts in these subdirectories fail since the anonymous user account by default does not have permissions that would allow it to start or stop services on Microsoft Windows 2000 or make changes to the Host Integration Server 2009 system. The subdirectories that must have IIS security enabled are as follows:

- SNASebAdmin\Change
- SNASebAdmin\Connections
- SNASebAdmin\Services
- SNASebAdmin\Status

It is possible to host these ASP pages on a computer running the Web server that is different from the computer running Host Integration Server 2009. However, this requires some changes to the ASP pages to handle connections to a different computer, security, and authentication issues.

See Also

Other Resources

Administration and Management Samples

VBScript ImportExport Sample

The Administration\VBScript folder contains a sample written in Microsoft Visual Basic Scripting Edition (VBScript) that illustrates how to import and export SNA configuration information from Host Integration Server 2009 in managed object format (MOF) using Windows Management Instrumentation (WMI). This sample relies on the MOFCOMP.exe application supplied with Microsoft Windows for importing.

In the following examples, ImportExport.VBS has been renamed to HISCFG.vbs.

The usage for this command-line tool for exporting configuration information is as follows:

```
HISCFG [/S:server] [/N:namespace] [/C:class] [/O:outfile]
       [/U:username] [/W:password] [/Q]
```

The various command-line options are explained in the following table.

Note
Case is ignored for command-line options except for help and either the '/' or '-' character is interpreted as the leading chara cter for an option. The following table uses the '/' character for illustration.

Command-Line Switch	Comments
/?	This flag shows the usage for this command and exits.
/C	The name of the WMI parent class to be queried. This should be set to one of the WMI classes defined in the MOF files supplied with Host Integration Server 2009. This parameter defaults to *MsSna_Config* and exports all of the classes SNA classes and their associations.
/h	This flag shows the usage for this command and exits.
/N	The WMI namespace to be queried. This parameter defaults to "root\MicrosoftHIS" for Host Integration Server 2009.
/O	The name of the file used for output. This parameter has no default value.
/Q	This Boolean flag indicates whether this is query should be completed quietly without displaying any status or error messages. This parameter defaults to verbose option.
/S	The name of the computer running Host Integration Server 2009. This parameter has no default value.
/U	The user name of a user on the domain or active directory where Host Integration Server 2009 is running wit h administrative rights. This parameter has no default value.
/W	The password of a user on the domain or active directory where Host Integration Server 2009 is running with administrative rights. This parameter has no default value.

The usage for this command-line tool for importing SNA configuration information is as follows:

```
HISCFG [/I:inputfilename]
```

A potential problem using WMI can occur with duplicate logical unit (LU) pools that can be illustrated using this sample program. Normally, exporting and re-importing the MOF file would not create duplicates. However, the HIS WMI provider allows pool-to-workstation association instances to be duplicated because, by design, duplicates of this type of object are allowed. It is possible to associate the same pool to the same workstation or user multiple times. Emulators use this to create more sessions for clients. Therefore, it is not possible to identify one such association from another. The WMISNA Provider,

WMISNA.DLL, always create new associations of these types, even if an association with the same pair (Pool, Wks) already exists. Only in the case of this object type is this allowed. However, this can create a problem for applications developed using WMI (the Import/Export sample, for example) if the application does not know to not create the duplicates.

The following sequence illustrates this issue using the ImportExport sample:

1. Create a Pool-Workstation association by using Host Integration Server Manager or the Administration Manager client.
2. Export the SNA configuration to a MOF file using the ImportExport utility.
3. Import that same MOF file again using the ImportExport utility.
4. Duplicate Pool-Workstation associations will be created.

The result is that if a client uses the import/export sample or a similar application developed using WMI on a Host Integration Server 2009 configuration that has pool-to-workstation associations, then the number of associations will effectively double after running the sample. The workaround using the ImportExport sample would be as follows:

1. Export the configuration to a MOF file.
2. Remove the pool to workstation associations from the MOF file just created.
3. Import the MOF file back.

When importing the configuration from one domain to another using the ImportExport sample or a similar application developed using WMI, then step 2 should be ignored. Normally, WMI applications should copy an existing configuration to a blank configuration file so this condition does not arise.

See Also

Other Resources

Administration and Management Samples

APPC Samples

The Host Integration Server 2009 SDK includes the source code for several sample programs that illustrate using Advanced Program-to-Program Communications (APPC). These sample programs are copied to your hard drive during Host Integration Server software or Host Integration Client software installation when the **Host Integration Server Software Development Kit** option is selected. These samples are installed in the Samples\NetworkIntegration\APPC subdirectory below where the Host Integration Server SDK software is installed (C:\Program Files\Microsoft Host Integration Server\SDK, by default).

These APPC sample programs include the following:

APPC TP samples	Description
APPC Send and Receive TPs	Sample programs in C that represent simple APPC send and receive transaction programs (TPs) illustrating the use of asynchronous verb completion. The samples located in the SENDRECV subdirectory implement simple bulk data sending and receiving TPs (SENDTP and RECVTP). These samples are located in the \APPC\SendRecv subdirectory on the CD.
Multithreaded Send and Receive TPs	Sample programs in C that represent more advanced APPC send and receive TPs illustrating the use of multiple threads and multiple conversations per thread. The multithreaded receive TP samples illustrate using events or IO completion ports for notification These samples are located in the \APPC\msendrcv subdirectory on the CD.

In addition to these APPC sample programs, the following supplemental programs are included on the Host Integration Server CD-ROM.

Supplemental program	Description
TPSETUP	A sample installation program in C demonstrating an interface that assists in the configuration of autostarted invokable TPs. This sample is located in the \APPC\tpsetup subdirectory on the CD.
TPSTART	A sample program in C required for the automatic startup of invokable TPs that run as applications under Microsoft Windows 2000. TPSTART is not required if a TP is written as a Windows 2000 service. An executable binary of TPSTART is installed by Host Integration Server Setup in the SYSTEM subdirectory of the Host Integration Server root directory. This sample is located in the \APPC\tpstart subdirectory on the CD.

In This Section

Building the TPs

TPSETUP

TPSTART

TPSTART.ini

APPC Send and Receive TPs

Multithreaded Send and Receive TPs

Reference

APPC Programmer's Reference

Related Sections

APPC Programmer's Guide

See Also

Other Resources

Network Integration Samples

Building the TPs

The APPC samples are designed to be built by using the command-line compiler or the interactive development environment (IDE) in Microsoft Visual Studio .NET 2003 or later.

To build the APPC samples installed from the Host Integration Server CD, set the following environment variables:

Variable	Specifies
ISVLIBS	Directory containing the Microsoft Host Integration Server LIB files for Microsoft Windows 2000
ISVINCS	Directory containing the WINSNA header files
SAMPLEROOT	Root directory of the sample code

For example, if you installed the Host Integration Server SDK directory to the default location (C:\Program Files\Microsoft Host Integration Server\SDK), use the following lines to set the variables (assuming that Intel binaries are being produced for Windows 2000):

```
ISVLIBS=C:\Program Files\Microsoft Host Integration Server\SDK\LIB
ISVINCS=C:\Program Files\ Microsoft Host Integration Server \SDK\Include
SAMPLEROOT=C:\Program Files\ Microsoft Host Integration Server \SDK\Samples\SNA
```

To build a specific sample (SendTp, for example) using the Visual Studio IDE, start Visual Studio and open the appropriate Microsoft Visual C++ project file (NetworkIntegration\appc\sendtp.vcproj, for example) from the **File** menu. Select a configuration and build the sample from the **Build** menu. Each Visual C++ project file has two configurations, one for a DEBUG build and one for a RETAIL build.

See Also

Other Resources

APPC Samples

TPSETUP

TPSETUP is a program that simplifies the setting of registry or environment variables needed by autostarted invokable TPs. Without an interface like that provided by TPSETUP, configuring such variables can be complicated and error prone. Therefore, it is recommended that you use code like TPSETUP in installation programs for autostarted invokable TPs.

Operation

INSTALL.C, the source code for TPSETUP, can be compiled to work in Microsoft® Windows® 2000.

It is recommended that autostarted invokable TPs be written as Windows 2000 services. To create the installation program for such TPs, study the code in INSTALL.C. For example, use the **CreateService** function or similar code when installing a TP that will run as a service under Windows 2000. (For important information about how services work under Windows 2000, see the documentation for Windows 2000.)

See Also

Other Resources

APPC Samples

TPSTART

An autostarted TP that runs as an application under Microsoft® Windows® 2000 requires the support of the TPSTART program, which is installed with the Microsoft Host Integration Server software in the SYSTEM subdirectory of the Host Integration Server root directory. Therefore, the TPSTART program must be started on a Windows 2000-based client before an autostarted invokable TP can be started as an application. You can start TPSTART by using standard Windows 2000 methods, such as including TPSTART in the **Startup** group on the client.

See Also

Other Resources

APPC Samples

TPSTART.ini

You can use TpStart.ini to specify custom tracing options when using TPSTART.exe.

The TpStart.ini file should be placed in the <snaroot> folder.

TpStart.ini sample

```
[TpStart]
HIDE=0
QSIZE=64
TRACE=1
FILE=C:\traces\tpstart.txt
LEVEL=0
FSIZE=2000000

Details for the entries are:

HIDE - 1: Hides the icon
            0: Shows the icon

QSIZE - Size of the pending queue, MIN=16,MAX=256
             Note, if QSIZE is not included in the ini file, a size of 16 is used

TRACE - 1: Enable tracing
               0: Disable tracing

FILE - The path and filename to be created for the trace, note the folder must already exis
t

LEVEL - The level of tracing (0 - 10):  0 - the most detailed tracing,  10 - no tracing

FSIZE - Maximum file size, MIN=10000, MAX=0xFFFFFFFF
             Note, if FSIZE is not included in the ini file, a size of 10000 is used.
```

For additional information about TpStart.exe, see KB article 137074 at http://support.microsoft.com/kb/137074.

APPC Send and Receive TPs

These are simple APPC send and receive TPs that illustrate the use of asynchronous verb completion. This sample implements simple bulk data sending and receiving TPs (SENDTP and RECVTP).

Setup

To set up these TPs, create an appropriate APPC LU-LU-mode triplet. The default is SENDLU-RECVLU-#INTER, but this can be configured (see the following sections).

Input and Output

The APPC send and receive TPs each use a configuration file for input. To name the file, use .CFG as the extension, and use the same base file name as the TP executable file (SENDTP.CFG, for example). Save this configuration file in the same directory location as the executable file (the TP itself).

For SENDTP, the configuration file (called SENDTP.CFG if the executable file is SENDTP.EXE) can contain the following items, one per line, in any order. If a variable is not found in the file or the file is not present at all, the default is used.

Line	Default value	Value to supply
ResultFile =	C:\SENDTP.OUT	File name to print timings to
LocalLUAlias =	SENDLU	Local LU alias
RemoteLUAlias =	RECVLU	Remote LU alias
ModeName =	#INTER	Mode name
LocalTPName =	SENDTP	Name of local TP
RemoteTPName =	RECVTP	Name of remote TP
NumConversations =	1	Number of conversations
NumSends =	2	Number of SEND_DATA verbs per conversation
SendSize =	1024	Size in bytes of data sent each time
ConfirmEvery =	1	Number of **SEND_DATA** verbs between CONFIRM verbs
SendConversation =	No	Yes or No: Use the SEND_CONVERSATION verb rather than the sequence of ALLOCATE, **SEND_DATA**, DEALLOCATE

RECVTP uses a RECVTP.CFG file in a similar way, but only to read the **LocalTPName** field.

The output from SENDTP and RECVTP consists of details of the configuration and the time taken for each conversation, and is sent to the result file specified in SENDTP.CFG.

Operation

RECVTP should be started first; it issues RECEIVE_ALLOCATE with the specified TP name. SENDTP is then started; it first issues MC_SEND_CONVERSATION to tell RECVTP how many conversations will be carried out. It then carries out the specified number of conversations.

For SENDTP, each conversation consists of an MC_ALLOCATE verb, followed by a given number of MC_SEND_DATA verbs of a given size, and interspersed with MC_CONFIRM verbs at a given interval, followed by an MC_DEALLOCATE.

RECVTP issues MC_RECEIVE_AND_WAIT when **RECEIVE_ALLOCATE** completes, and then issues either MC_RECEIVE_AND_WAIT or MC_CONFIRMED according to the return from the previous **MC_RECEIVE_AND_WAIT**.

At any stage, if the TPs encounter an error, they terminate. Use APPC API tracing to diagnose problems with the configuration.

At the end of the specified number of conversations, SENDTP sends timing information to a file.

Both TPs are built from a single source code file, SENDRECV.C. SENDTP is compiled only if **-DSENDTP** is used on the command line.

The TPs run as Microsoft® Windows® 2000 applications with a minimized window, the title bar of which displays the status. When the WndProc of this window, TPWndProc, receives the WM_CREATE message for the window, this triggers the issuing of the first verb. When TPWndProc receives an ASYNC_COMPLETE message from Windows APPC, this triggers the issuing of the next verb, dependent on what the previous verb was. When the window is closed, WinAPPCCleanup is issued to terminate any active conversations.

See Also

Other Resources

APPC Samples

Multithreaded Send and Receive TPs

These multithreaded send and receive TPs are more advanced than the single-threaded equivalents. The samples located in the MSENDRCV subdirectory all use the asynchronous interface of APPC, with verb completion signaled by events (WinAsyncAPPCEx) or IO completion ports (WinAsyncAPPCIOCP). These TPs show how to code multithreaded APPC applications with multiple conversations per thread. They are more complex than the single-threaded equivalents, but are also more realistic.

If you are unfamiliar with APPC, examine the single-threaded TPs first. If you are unfamiliar with methods of creating threads or processing events in Microsoft® Windows® 2000, see the Microsoft Platform SDK documentation along with the multithreaded TPs.

Setup

There are four multithreaded send and receive routines that illustrate using asynchronous APPC calls:

- MRCV for receiving using events for notification
- MRCVIO for receiving using IO completion ports for notification
- MSEND for sending using events for notification
- MSENDRCV for simultaneous sending and receiving using events for notification

To set up these TPs, create an appropriate APPC LU-LU-mode triplet. The default is SENDLU-RECVLU-#INTER, but this can be configured (see the sections that follow). To run a large number of simultaneous conversations, increase the session limits for #INTER or use another mode with large session limits.

One obvious way of configuring these programs is to configure MSEND to run with MRCV or MRCVIO; another way is to configure MSENDRCV to run with another copy of MSENDRCV. However, you can also configure MSEND to run with one or more copies of RECVTP (the single-threaded version) and MRCV or MRCVIO to run with one or more copies of SENDTP. You can also configure MSENDRCV to run with MSEND, MRCV, SENDTP or RECVTP. For more information, see the sections that follow.

One possible arrangement is to place SENDTP (single-threaded) on multiple client computers, and configure MRCV or MRCVIO (multithreaded) on a server so that it interacts with all the TPs on the clients. Many other arrangements are possible.

Configuration for MRCV, MSEND, and MSENDRCV

The MRCV, MSEND, and MSENDRCV TP samples use a configuration file for configuration and input. To name the file, use .CFG as the extension, and use the same base file name and directory location as the executable file (the TP itself).

The following table shows examples of CFG files that could be used with MSEND and MRCV.

Example of MSEND.CFG file	Example of MRCV.CFG file
ResultFile=MSEND.OUT	TraceFile=MRCV.TRC
TraceFile=MSEND.TRC	LocalTPName=MRCVTP
RemoteTPName=MRCVTP	NumRcvConvs=32
LocalLUAlias=LUA	NumRcvThreads=4
RemoteLUAlias=LUB	RcvSize=4096
ModeName=#INTER	
NumSendConvs=32	

NumSends=128	
ConfirmEvery=16	
SendSize=256	

For MSEND, the configuration file (MSEND.CFG) can contain the following items, one per line, in any order. If a variable is not found in the file or the file is not present, the default is used.

Line	Default value	Value to supply
ResultFile =	MSEND.OUT	File name to print timings to (located in default directory for MSEND)
TraceFile =	MSEND.TRC	Trace file name (located in default directory for MSEND)
LocalLUAlias =	SENDLU	Local LU alias
RemoteLUAlias =	RECVLU	Remote LU alias
ModeName =	#INTER	Mode name
RemoteTPName =	MRCVTP	Name of remote TP (for MC_ALLOCATE)
NumSendConvs =	4	Number of conversations to send
NumSends =	8	Number of MC_SEND_DATA verbs per conversation
SendSize =	256	Size in bytes of data sent each time
ConfirmEvery =	2	Number of **MC_SEND_DATA** verbs between MC_CONFIRM verbs

The following lines are for MRCV:

Line	Default value	Value to supply
TraceFile =	MRCV.TRC	Trace file name (located in default directory for MSEND)
LocalTPName =	MRCVTP	Name of local TP (for RECEIVE_ALLOCATE)
NumRcvConvs =	4	Number of conversations to receive
NumRcvThreads =	2	Number of threads to start for processing receive conversations
RcvSize =	4096	Size in bytes of receive buffer for MC_RECEIVE_AND_WAIT

The following table shows examples of configuration files (MSENDRCV.CFG) that could be used with MSENDRCV. Each row of the table (Example A and Example B) contains two files that work together on a pair of computers.

Example A of MSENDRCV.CFG	Example B of MSENDRCV.CFG
ResultFile=MSENDRCV.OUT	ResultFile=MSENDRCV.OUT
TraceFile=MSENDRCV.TRC	TraceFile=MSENDRCV.TRC
LocalTPName=TPA	LocalTPName=TPB
RemoteTPName=TPB	RemoteTPName=TPA

LocalLUAlias=LUA	LocalLUAlias=LUB
RemoteLUAlias=LUB	RemoteLUAlias=LUA
ModeName=#INTER	ModeName=#INTER
NumRcvConvs=50	NumRcvConvs=25
NumRcvThreads=4	NumRcvThreads=4
RcvSize=4096	RcvSize=4096
NumSendConvs=25	NumSendConvs=50
NumSends=100	NumSends=100
ConfirmEvery=10	ConfirmEvery=10
SendSize=256	SendSize=256

The following lines are for MSENDRCV:

Line	Default value	Value to supply
ResultFile =	MSENDRCV.OUT	File name to print timings to (located in default directory for the MSEND or MSENDRCV sending TP)
TraceFile =	MSENDRCV.TRC	Trace file name (located in default directory for the MSEND or MSENDRCV sending TP)
LocalLUAlias =	SENDLU	Local LU alias
RemoteLUAlias =	RECVLU	Remote LU alias
ModeName =	#INTER	Mode name
RemoteTPName =	MRCVTP	Name of remote TP (for MC_ALLOCATE)
NumSendConvs =	4	Number of conversations to send
NumSends =	8	Number of MC_SEND_DATA verbs per conversation
SendSize =	256	Size in bytes of data sent each time
ConfirmEvery =	2	Number of **MC_SEND_DATA** verbs between MC_CONFIRM verbs
LocalTPName =	MRCVTP	Name of local TP (for RECEIVE_ALLOCATE)
NumRcvConvs =	4	Number of conversations to receive
NumRcvThreads =	2	Number of threads to start for processing receive conversations

RcvSize =	4096	Size in bytes of receive buffer for MC_RECEIVE_AND_WAIT

The output from MSEND and MSENDRCV consists of details of the configuration and the time taken for each conversation, and is sent to the result file specified in MSEND.CFG or MSENDRCV.CFG.

Operation of MRCV, MSEND, and MSENDRCV

The MRCV, MSEND, and MSENDRCV TPs use multiple event processing in Windows 2000 to avoid creating an unnecessary number of threads.

These TPs also use Windows-based processing, but this is incidental. Its only purpose is to display beneath the icon on the screen a running count of threads, the number of conversations currently sending or receiving data, and the number of conversations completed. The Windows-based processing could easily be removed to create a completely batch-oriented program. To do this, termination would need to be signaled with an event rather than with WM_CLOSE.

The TP name used in TP_STARTED is the name of the executable file (MSEND, MRCV, or MSENDRCV). The TP names used in MC_ALLOCATE and RECEIVE_ALLOCATE can be configured, as shown in the preceding tables.

MSEND reads its configuration file (or uses defaults) to determine the number of send conversations to start. Each conversation reads the value of **NumSends** (or uses the default), issues that number of MC_SEND_DATA verbs, and then terminates. When all of the conversations for a thread have terminated, the thread itself terminates. When all of the send threads have terminated, the program terminates.

An MC_CONFIRM verb is issued before the first **MC_SEND_DATA** and then at the intervals specified by **ConfirmEvery**. The complete data flow for a conversation is as follows:

TP_STARTED

MC_ALLOCATE

MC_CONFIRM

MC_SEND_DATA (repeated the number of times specified by **ConfirmEvery**)

MC_CONFIRM

MC_SEND_DATA (repeated the number of times specified by **ConfirmEvery**)

MC_CONFIRM

(Pattern repeats until the number of **MC_SEND_DATA** verbs equals **NumSends**.)

MC_DEALLOCATE

TP_ENDED

MRCV starts up an initial thread for issuing RECEIVE_ALLOCATE verbs, and then reads its configuration file (or uses defaults) to determine the number of receive threads to start and the number of conversations to receive. The initial thread issues a **RECEIVE_ALLOCATE** and waits. When the **RECEIVE_ALLOCATE** completes, the initial thread turns the processing of the conversation over to the next available receive thread, and issues another **RECEIVE_ALLOCATE**. This process continues until the configured number of **RECEIVE_ALLOCATE** verbs (that is, **NumRcvConvs**) have completed.

There is a limit to the number of conversations that can be supported on a thread, because of the limit to the number of events that can be waited for with **WaitForMultipleObjects** (a function in the Win32® API). For send threads, the limit is 64 conversations per thread; for receive threads, the limit is 63 conversations per thread.

MSEND works with this limit by starting enough threads to support the configured number of conversations. For example, if **NumSendConvs** is set to 200, four send threads are started: three of them process 64 conversations each and one processes the remaining eight conversations.

MRCV works with this limit by comparing **NumRcvConvs** to **NumRcvThreads**. If **NumRcvConvs** is more than (63 * **NumRcvThreads**), **NumRcvThreads** is increased. If **NumRcvThreads** is greater than **NumRcvConvs**, **NumRcvThreads** is reduced to prevent creating unneeded threads.

With MRCV, to ensure that a receive thread correctly picks up the conversation, two special events are used per thread: *event1* and *event2*. The following table illustrates their use.

RECEIVE_ALLOCATE thread	**Receive thread**
Issue RECEIVE_ALLOCATE and wait	Wait for *event1*

(**RECEIVE_ALLOCATE** completes)	
Select next receive thread and set *event1* for that thread; then wait for *event2* for that thread	
	(*Event1* completes)
	Add conversation to list of conversations being processed
	Set *event2*
(*Event2* completes)	
REPEAT	REPEAT

The receive thread waits not only on the *event1* set for it, but also on one event for each conversation the thread is processing.

If **NumRcvConvs** is set to zero, the **RECEIVE_ALLOCATE** thread will never terminate. If **NumSends** is set to zero, the conversation will never terminate; this is useful for getting the maximum number of simultaneous conversations.

Tracing of MRCV, MSEND, and MSENDRCV

If you want to observe the detailed processing of the MRCV, MSEND, or MSENDRCV sample TPs, you can enable tracing. To do this, find the following line, commented out, near the top of the file:

```
#define SRTRC
```

Enable this line or define this value on the command-line option to the compiler, and trace statements will be written to the trace file(s) specified by the **TraceFile** variable in the configuration.

There are also some trace statements that have been commented out. If they are left commented out, only MC_CONFIRM and MC_CONFIRMED processing is traced while a conversation is running, to maintain a send or receive count without generating a large amount of trace information. You can activate the detailed tracing of events (such as the sending of data) by enabling one or more trace statements.

The Trace Initiator (snatrace.exe) tool provides APPC API tracing for Microsoft Host Integration Server applications. For more information about the Trace Initiator and Trace Viewer tools, see Microsoft Host Integration Server Help.

Configuration for MRCVIO

The MRCVIO TP sample is a multithreaded console application that uses command-line options for configuration and input. If an option is not provided on the command line, then the default is used. The following table lists the possible command-line options for MRCVIO.

Command-line option	**Description**
-?	Displays usage information for this sample and exits.
-c numRcvConvs	The maximum number of APPC conversations to support. The default value is 8 with a maximum value of 64.
-d duration	The number of seconds that the sample application should run. The default value is 60 seconds. A value of 0 for duration means run indefinitely.
-h	Displays usage information for this sample and exits.

-i IntTraceFile	The name of the internal trace file if tracing is required. When this command-line option is specified, internal tracing is enabled. This option has no default value and internal tracing is turned off.
-n numRcvThreads	The number of Completion Port threads to allocate. The default value is 4 with a maximum value of 32.
-r rcvSize	The size in bytes of the buffer supplied on each **RECEIVE_ALLOCATE**. The default value is 4096 with a maximum value of 65535.
-t TPName	The name supplied on the **RECEIVE_ALLOCATE** verbs. The default value is the "MRCVTP" string.

Operation of MRCVIO

The MRCVIO TP sample uses IO completion ports for notification and will only operate on Windows 2000. Using the IO completion port mechanism is the preferable method for writing scalable APPC server applications.

The MRCVIO TP sample contains the routines for a multithreaded console application that uses asynchronous APPC calls on a single I/O completion port to receive data. The MRCVIO sample creates a small pool of threads that will be used for processing RECEIVE requests. It operates in conjunction with one of the following:

- Single-threaded version of send(SENDTP)
- Multithreaded event-based versions of send (MSEND, MSENDRCV)

The MRCVIO sample uses a server model that continues to accept conversations via RECEIVE_ALLOCATE until the application is manually terminated, or a specified timer expires. The conversations do not belong to any particular RECEIVE thread. Each receive thread issues **GetQueuedCompletionStatus** calls to wait for completion of an APPC verb (on any conversation). Each conversation issues **MC_RECEIVE_AND_WAIT** verbs to receive data. If confirmation is requested, an **MC_CONFIRM** verb is issued.

The TP name used in MC_ALLOCATE and RECEIVE_ALLOCATE can be configured by using the command-line options as shown in the preceding table of options for the configuration of MRCVIO.

MRCVIO starts up and parses its command-line options (or uses defaults) to determine the number of receive threads to start, the number of conversations to receive, the buffer size for each **RECEIVE_ALLOCATE**, its TP name, how long the application should run, and whether internal tracing is enabled.

Once command-line options are parsed, the MRCVIO sample calls the **CreateCompletionPort** function to allocate the IO completion port. If this call is successful, then the specified number of threads are created with the thread start routine pointing to the MRCVIO **ReceiveThread** function and the thread priority for each thread is lowered. Then the specified number of APPC conversations are started.

The MRCVIO sample uses the IO completion port structure and the WinAsyncAPPCIOCP function as listed below.

```
/* IOCP - Structure and function prototype          */
typedef struct
{
    HANDLE       APPC_CompletionPort;
    DWORD        APPC_NumberOfBytesTransferred;
    DWORD        APPC_CompletionKey;
    LPOVERLAPPED APPC_pOverlapped;
} APPC_IOCP_INFO;
extern HANDLE WINAPI WinAsyncAPPCIOCP(
    APPC_IOCP_INFO* iocp_handle, // IO completion port information
    long lpVcb);                 // pointer to APPC verb control block
```

The APPC_CompletionPort must be a HANDLE returned by the **CreateIoCompletionPort** function issued by the application before using **WinAsyncAPPCIOCP**. The other three fields can be set to any value whatsoever. The APPC library does nothing with these other fields, except to return them unaltered on the **GetQueuedCompletionStatus** when the APPC verb

completes. Application developers can set these values to whatever they like, but assuming the server application is handling multiple concurrent APPC conversations, an application will need to use one of these three fields to correlate APPC verbs with their completions.

For example, the MRCVIO sample passes a pointer to a Conversation Control Block into the APPC_pOverlapped field when it issues an APPC verb. The same value is returned when the APPC verb completes on the **GetQueuedCompletionStatus**. This allows the sample MRCVIO TP to figure out which APPC verb has actually completed. APPC developers can use a different method (an index into an array of VCBs, for example) to provide the same effect.

Also, an application might use the APPC_CompletionKey field to distinguish between APPC events and other events posted to this IO Completion port. For example, the MRCVIO sample sets this value to a user-defined constant IOCP_VERB_COMPLETE so that the **GetQueuedCompletionStatus** function can distinguish APPC verb completions from the other events that are posted to this IO Completion port (IOCP_START_CONVERSATION, IOCP_END_CONVERSATION and IOCP_TERMINATE_THREAD). However, this is purely for the convenience of the application. An APPC developer could decide not to post any events to its IO Completion port (except implicitly for APPC completions). In such a case, it would be unnecessary to set any value in the APPC_CompletionKey.

See Also

Other Resources

APPC Samples

CPI-C Samples

Host Integration Server 2009 SDK includes the source code for several sample programs that illustrate using Common Programming Interface for Communications (CPI-C) for transaction programs (TPs).

These files are copied to your hard drive during Host Integration Server software or Host Integration Client software installation when the Host Integration Server Software Development Kit option is selected. These samples are installed in the Samples\NetworkIntegration\CPI-C subdirectory below where the Host Integration Server SDK software is installed (C:\Program Files\Microsoft Host Integration Server\SDK, by default).

These sample programs include the files listed in the following table.

Sample TP program	Description
APING and APINGD	Sample programs that provide a simple test for end-to-end connectivity. These samples are located in the \CPI-C\aping folder on the CD.
Multithreaded APINGD	A multithreaded connectivity test that illustrates nonqueued behavior in Microsoft Windows Server 2003 and Windows 2000. This sample is located in the \CPI-C\mping folder on the CD.
CPI-C Send and Receive TPs	A pair of simple CPI-C TPs that illustrate the use of asynchronous CPI-C calls. These samples are located in the \CPI-C\CpicSendRecv folder on the CD.
AREXEC and AREXECD	A pair of TPs that execute commands on a remote computer and send the output back across the connection. These samples are located in the \CPI-C\ArExec folder on the CD.
AREMOTE	A sample client and server program using APPC that enables you to invoke and control a text-mode program from another computer. This sample using APPC was based a Win32 sample program that originally used named pipes. These samples are located in the \CPI-C\ARemote folder on the CD.

In addition to these TPs, the following supplemental programs are included on the Host Integration Server 2009 CD.

Supplemental program	Description
TPSETUP	A sample installation program, demonstrating an interface that assists in configuring autostarted invokable TPs. This sample is located in the \NetworkIntegration\APPC\tpsetup folder on the CD.
TPSTART	A program required for the automatic startup of invokable TPs that run as applications under Windows Server 2003 or Windows 2000. TPSTART is not required if the TP has been written as a Windows Server 2003 or Windows 2000 service.TPSTART is installed by Host Integration Server Setup in the System folder of the Host Integration Server root directory. This sample is located in the \NetworkIntegration\APPC\tpstart folder on the CD.

In This Section

Building the TPs

APING and APINGD

Multithreaded APINGD

CPI-C Send and Receive TPs

AREXEC and AREXECD

AREMOTE

Reference

CPI-C Programmer's Reference.

Related Sections

CPI-C Programmer's Guide.

See Also

Other Resources

Network Integration Samples

Building the TPs

The Common Programming Interface for Communications (CPI-C) samples are designed to be built using the command-line compiler or the IDE in Microsoft Visual Studio.

To build the CPI-C samples installed from the Host Integration Server CD, set the following environment variables listed in the following table.

Variable	Description
ISVLIBS	The directory containing the Host Integration Server 2009 LIB files for Microsoft Windows Server 2003 or Windows 2000.
ISVINCS	The directory containing the Host Integration Server 2009 header files.
SAMPLEROOT	The root directory where the sample code provided as part of the SDK has been installed on a local hard disk.

For example, if you installed the Host Integration Server SDK directory to the default location (C:\Program Files\Microsoft Host Integration Server\SDK), use the following lines to set the variables (assumes Intel binaries are being produced for Windows Server 2003 or Windows 2000.

```
ISVLIBS=C:\Program Files\Microsoft Host Integration Server\SDK\LIB
ISVINCS=C:\Program Files\Microsoft Host Integration Server\SDK\Include
SAMPLEROOT=C:\Program Files\Microsoft Host Integration Server\SDK\Samples\SNA
```

To build a specific sample (APING, for example) using Visual Studio .NET 2003, open the appropriate Microsoft Visual C++ project file (SNA\aping\aping.vcproj, for example) from the **File** menu. Select a configuration and build the sample from the **Build** menu. Each project file has two configurations, one for a DEBUG build and one for a RETAIL build.

See Also
Other Resources
CPI-C Samples

APING and APINGD

The sample code for APING and APINGD is ported from code on the IBM Advanced Program-to-Program Communications (APPC)/Common Programming Interface for Communications (CPI-C) disk. These samples are used to test end-to-end connectivity, and simply show that the configuration is correct by exchanging bytes of data across the link. APING is the client or invoking (local) half and attempts to contact APINGD (the APPC/CPI-C ping daemon or server), which is written here as a Microsoft Windows Server 2003 or Windows 2000 service so it can be installed as an invokable TP on the remote computer.

Setup

To use the APING and APINGD samples included in the Microsoft Host Integration Server 2009 SDK, follow these steps:.

To set up APING and APINGD

1. Create an appropriate APPC LU-LU-mode triplet (for example, LUPING-LUPINGD-#INTER).
2. Set up a CPI-C symbolic destination name that contains the configured remote LU and mode. (The TP name for APINGD is APINGD.)
3. Assign the local APPC LU to the APING TP, either by using a registry entry of APING:REG_SZ:*LocalLUAlias* in the **SnaBase\Parameters\Clients** key, or by assigning the local LU as the default local APPC LU for the user who will run APING.

Input and Output

APING is a console application. The syntax of its command line is

aping [**-s***size*] [**-i***iterations*] [**-c***packets*] [**-m***mode*] [**-t***tpname*] *PartnerLUName*

aping [**-s***size*] [**-i***iterations*] [**-c***packets*] *SymbolicDestinationName*

where

-s *size*

Specifies the size, in bytes, of the packet transmitted. The default is 100 bytes.

-i *iterations*

Specifies the number of iterations to carry out. The default is 2.

-c *packets*

Specifies the number of consecutive packets sent by each side. The default is 1.

-m *mode*

Specifies the mode name. The default is #INTER.

-t *tpname*

Specifies the TP name of the TP to start on the remote server. The default is APINGD.

PartnerLUName

Specifies the partner LU name of the destination.

SymbolicDestinationName

Specifies the symbolic destination name of the destination.

Output goes to **stdout** and **stderr**, and details the data rates and timings for each iteration.

Operation

Note that with APINGD, Specify_Local_TP_Name is used to set the local TP name, so Wait_For_Conversation must be used to wait for the Accept_Conversation call to complete, because it will return asynchronously.

The code at the end of APINGD.C is a stub for making any TP into a Windows Server 2003 or Windows 2000 service. There are three routines that are needed: **main**, **ServiceMain**, and **ControlHandler**. For details about how these work, see the comments in the file. The **TPStart** routine is the entry point of the TP proper.

In particular, note that in response to the SERVICE_CONTROL_STOP or SERVICE_CONTROL_SHUTDOWN messages in the **ControlHandler** routine, action should normally be taken to stop the service, but because each run does not last long with these samples, no code is included to take such an action.

See Also

Other Resources

CPI-C Samples

Multithreaded APINGD

The version of MPINGD provided in the sample code illustrates how to achieve nonqueued behavior from an invokable TP in Microsoft® Windows Server™ 2003 and Windows® 2000. This means that multiple copies of APING can talk to the same copy of MPINGD at the same time. However, you cannot run multiple copies of a Windows Server 2003 or Windows 2000 service. The features are achieved by always having a thread with an Accept_Conversation outstanding, so that any incoming attach for MPINGD will always be satisfied immediately.

Setup

Setup requirements for MPINGD are the same as for the single-threaded version, APINGD. The remote LU and mode that you use should support parallel sessions so that more than one conversation at a time is possible.

Input and Output

The input and output for MPINGD are the same as for the single-threaded version, APINGD.

Operation

The operation of MPINGD is similar to that of the single-threaded version, APINGD. The same three routines are used (**main**, **ServiceMain** and **ControlHandler**). **ServiceMain** calls the **TPStart** routine. This routine must not return until the service is ready to terminate.

The **TPStart** routine does some initialization, creates the first conversation thread, and then waits on an event created by the **ServiceMain** routine. This event is set when the service control manager issues an order to STOP or SHUTDOWN. When the event is set, it calls WinCPICCleanup, which will cancel any active conversations and return outstanding Accept_Conversation calls, thus making all conversation threads exit. It then marks the service as STOPPED.

The **ThreadStart** routine is the entry point for each of the conversation threads. It issues **Accept_Conversation** and Wait_For_Conversation, and when this completes, it creates another thread to wait for the next attach while the existing thread services the first attach.

See Also

Other Resources

CPI-C Samples

CPI-C Send and Receive TPs

These transaction programs (TPs) are Common Programming Interface for Communications (CPI-C) versions of the APPC send and receive TPs. The sample code illustrates the use of asynchronous CPI-C calls.

Setup

In order to use these TPs, follow these steps:

To set up the send and receive TPs

1. Create an appropriate APPC LU-LU-mode triplet.
2. Set up a CPI-C symbolic destination name that contains the configured remote LU and mode. (The default symbolic destination name is CPICRECV.)
3. Assign the local APPC LU to the CPICSEND TP, either by using a registry entry of CPICSEND:REG_SZ:*LocalLUAlias* in the **SnaBase\Parameters\Clients** key, or by assigning the local LU as the default local APPC LU for the user who will run CPICSEND.

For example, use SENDLU-RECVLU-#INTER as your LU-LU-mode triplet. Then, create a CPI-C symbolic destination name CPICRECV containing the application TP name CPICRECV, the partner LU alias RECVLU, and the mode name #INTER. Finally, add the intended user to the users list, and assign SENDLU as the users default local APPC LU.

Input and Output

CPICSEND and CPICRECV use the files Cpicsend.cfg and Cpicrecv.cfg for input. These files should be placed in the folder that contains the TP executable file. These files are similar to the input files for the APPC send and receive TPs.

The following entries are for CPICSEND only:

Line	Default Value	Description
ResultFile =	C:\Cpicsend.out	The file name where the timings results will be stored.
NumSends =	2	The number of Send_Data calls per conversation.
SendSize =	1024	The size of data sent each time in bytes .
ConfirmEvery =	1	The number of **Send_Data** calls between Confirm calls. If ConfirmEvery=0, CPICSEND will not issue CONFIRM verbs.
SymDestName =	CPICRECV	The symbolic destination name.
NumConversations =	1	The number of conversations. This setting must be the same for CPICSEND and CPICRECV. (They do not negotiate the number.) If this value is zero, the TPs will do an infinite number of conversations.

WaitMode=	No	Yes, No, or Block. If WaitMode=No, the verbs are completed through posted windows messages. The TPs issue Specify_Windows_Handle with a window handle so that Windows CPI-C will post completion messages to this window handle. If WaitMode=Yes, verbs are non-blocking and completed using asynchronous call completion. In this case, the TPs issue **Specify_Windows_Handle** with NULL so that the TPs must then issue a Wait_For_Conversation call to wait for the asynchronous call to complete. If WaitMode=Block, all verbs are blocking.

The following entries are for CPICRECV only:

Line	**Default value**	**Description**
ResultFile =	C:\Cpicrecv.out	The file name where the timings results will be stored.
LocalTP Name =	CPICRECV	The local TP name to use on the Specify_Local_TP_Name call.
NumConversations =	1	The number of conversations. This setting must be the same for CPICSEND and CPICRECV. (They do not negotiate the number.) If this value is zero, the TPs will do an infinite number of conversations.
WaitMode=	No	Yes, No, or Block. If WaitMode=No, the verbs are completed through posted windows messages. The TPs issue Specify_Windows_Handle with a window handle so that Windows CPI-C will post completion messages to this window handle. If WaitMode=Yes, verbs are non-blocking and completed using asynchronous call completion. In this case, the TPs issue **Specify_Windows_Handle** with NULL so that the TPs must then issue a Wait_For_Conversation call to wait for the asynchronous call to complete. If WaitMode=Block, all verbs are blocking.

As with CPICSEND, CPICRECV produces C:\Cpicrecv.out (by default) with timings of the conversations in it.

Operation

CPICRECV should be started first. CPICRECV issues Specify_Local_TP_Name to set its local TP name, and then Accept_Conversation to accept a conversation (note that because **Specify_Local_TP_Name** is issued, the **Accept_Conversation** will complete asynchronously).

Both TPs issue Specify_Windows_Handle during initialization to set either the window handle or NULL. CPICSEND calls Set_Processing_Mode after completion of Initialize_Conversation to set the processing mode to non-blocking for this conversation.

After each call is issued, the return code is checked. If the return code is not CM_OPERATION_INCOMPLETE, the call has already completed, so an ASYNC_COMPLETE message is posted to trigger the next call. If **WaitMode** is set to YES and the issued call did not complete immediately, a Wait_For_Conversation call is issued to wait for call completion, at which point an ASYNC_COMPLETE message is posted. If **WaitMode** is set to NO and the issued call did not complete immediately, Windows CPI-C detects call completion and posts an ASYNC_COMPLETE message. The receipt of the ASYNC_COMPLETE message triggers the next call to be issued.

For CPICSEND, each conversation consists of an Allocate call, followed by a given number of Send_Data calls of given size and interspersed with Confirm calls at a given interval, followed by a Deallocate.

CPICRECV issues Receive on completion of the Accept_Conversation, and then issues either **Receive** or Confirmed according to the return from the previous **Receive**.

At any stage, if the TPs encounter an error, they terminate. Use CPI-C API tracing to diagnose problems with the configuration.

Both TPs are built from a single source-code file, CPICSR.C. CPICSEND is compiled only if CPICSEND macro is #defined. This macro is normally defined using the -DCPICSEND option on the command line to the C compiler.

The TPs run as Microsoft® Windows Server™ 2003 or Windows® 2000 applications with a minimized window, the title of which displays the status. When the WndProc of this window, TPWndProc, receives the WM_CREATE message for the window, it triggers the issuing of the first call. When TPWndProc receives an ASYNC_COMPLETE message from Windows CPI-C, it triggers the issuing of the next call, dependent on what the previous call was. When the window is closed, WinCPICCleanupis issued to terminate any active conversations.

See Also

Other Resources

CPI-C Samples

AREXEC and AREXECD

The sample code for these two transaction programs (TPs) provides the ability to execute commands on a remote computer and to send the output back across the connection to the invoking TP.

Setup

In order to use the AREXEC and AREXECD samples provided with the Microsoft Host Integration Server 2009 SDK, follow these steps:

To set up AREXEC and AREXECD

1. Create an appropriate APPC LU-LU-mode triplet.
2. Set up a CPI-C symbolic destination name that contains the configured remote LU and mode. (The TP name for AREXECD is AREXECD.)
3. Assign the local APPC LU to the AREXEC TP, either by using a registry entry of AREXEC:REG_SZ:*LocalLUAlias* in the **SnaBase\Parameters\Clients** key, or by assigning the local LU as the default local APPC LU for the user who will run AREXEC.

Input and Output

AREXEC is a console application. The syntax of its command line is

arexec [**-m***mode*] [**-t***tpname*] *destination command*

where

-m *mode*

Specifies the mode name. The default is #INTER.

-t *tpname*

Specifies the TP name.

destination

Specifies the destination. Can be either a symbolic destination name or a partner LU name.

command

Specifies the command string to execute on the remote computer.

The **stdout** and **stderr** from the command executed at the remote end is sent across the link and printed to **stdout** on the invoking end.

Operation

The AREXECD program is a Microsoft Windows Server 2003 or Windows 2000 service, using the same routines in APING, APINGD, and multithreaded APINGD. The execution of the command and sending back of data are done in the routine **execute_and_send_output** in CPICPORT.C. This sample creates a named pipe and connects to the read end of the pipe. It then creates a process to run the command and gives that process a handle to the write end of the pipe as its **stdout** and **stderr**. Then, the data is read from the pipe, and Send_Data is used to send it across the link.

See Also

Other Resources

CPI-C Samples

AREMOTE

The sample code for this TP provides the ability to control a text-mode program on a remote computer. AREMOTE is a Win32 console application that implements a client and a server. The AREMOTE server is invoked with the name of the text-mode program that the client wishes to control remotely. The AREMOTE client redirects **stdin** (keyboard input) from the client to the AREMOTE server. In turn, the AREMOTE server redirects **stdin** and **stderr** from the program being controlled back to the AREMOTE client.

Setup

In order to use the AREMOTE sample provided with the Host Integration Server 2009 SDK, follow these steps:

To set up AREMOTE

1. Create an appropriate APPC LU-LU-mode triplet.
2. Set up a CPI-C symbolic destination name that contains the configured remote LU and mode. (The TP name for AREMOTE is AREMOTE.)
3. Assign the local APPC LU to the AREMOTE TP, either by using a registry entry of AREMOTE:REG_SZ:*LocalLUAlias* in the **SnaBase\Parameters\Clients** key, or by assigning the local LU as the default local APPC LU for the user who will run AREMOTE.

Starting the client

The syntax of the command line to start the client end of AREMOTE is as follows.

```
        ServerLU [TPName] [TPName] [LocalLU]
    [Modename] [Lines] [Color] [Color]
```

Parameters

/C

Specifies the client mode.

ServerLU

Specifies the SNA LU for connecting to the server.

/T TPName

Specifies the TP name that the server is using. The default is AREMOTE.

/P TPName

Specifies the TP name that the client is using. The default is AREMOTE

/L LocalLU

Specifies the LU name for the local TP to use. The default is AREMOTE.

/M Modename

Specifies the mode name. The default is #INTER.

/N Lines

Specifies the number of lines to get.

/F Color

Specifies the foreground color. Color options are black, blue, green, cyan, red, purple, yellow, white, lblack, lblue, lgreen, lcyan, lred, lpurple, lyellow, and lwhite.

/B Color

Specifies the background color. Color options are black, blue, green, cyan, red, purple, yellow, white, lblack, lblue, lgreen, lcyan, lred, lpurple, lyellow, and lwhite.

Starting the server

The syntax of the command line to start the server end of AREMOTE is as follows.

```
Cmd [TPName] [Modename] [Color] [Color]
```

Parameters

/S

Specifies the server mode.

Cmd

Specifies a text-mode program that you want to control from another computer.

/T TPName

Specifies the TP name that the server is using. The default is AREMOTE.

/M Modename

Specifies the mode name. The default is #INTER.

/F Color

Specifies the foreground color. Color options are black, blue, green, cyan, red, purple, yellow, white, lblack, lblue, lgreen, lcyan, lred, lpurple, lyellow, and lwhite.

/B Color

Specifies the background color. Color options are black, blue, green, cyan, red, purple, yellow, white, lblack, lblue, lgreen, lcyan, lred, lpurple, lyellow, and lwhite.

Output

The **stdout** and **stderr** from the command run at the remote end is sent across the link and printed to **stdout** on the client. The **stdin** from the client is sent across the link and becomes the **stdin** for the command run at the remote end.

The APPC remote installer (ARSETUP) included with this sample brings up a dialog box that prompts for TP configuration information. The information is then placed in the registry under Microsoft® Windows Server™ 2003 or Windows® 2000. The WIN32 compiler flag specifies that the Win32 version of ARSETUP should be built for use on Windows Server 2003 or Windows 2000.

Notes

The AREMOTE server can also be configured to run as a Windows Server 2003 or Windows 2000 service using the ARSETUP sample utility included in the same folder on the CD.

The AREMOTE client can exit by inputting the following character sequences:

%cQ : Quit but leave the AREMOTE server running.

%cK : Exit and stop the AREMOTE server.

Other special client commands include the following:

%cM : Send a message to the AREMOTE server.

%cP : Show a popup on the AREMOTE server.

%cS : Report the status of the AREMOTE server.

%cH : Provide help describing these special client commands.

See Also

Other Resources

CPI-C Samples

LUA Samples

The source code for several sample programs that illustrate using logical unit application (LUA) are included in the Host Integration Server 2009 SDK.

These files are copied to your hard drive during Host Integration Server software or Host Integration Client software installation when the Host Integration Server Software Development Kit option is selected. These samples are installed in the Samples\SNA subdirectory below where the Host Integration Server SDK software is installed (C:\Program Files\Microsoft Host Integration Server \SDK\Samples\NetworkIntegration\LUA, by default).

These sample programs include the following files.

Sample transaction program (TP)	Description
RUI3270	Sample code for a simple emulator based on using the Request Unit Interface (RUI) API. Sample code is provided for Win32 applications. This sample is located in the LUA\RUI3270 folder on the CD.
SLI3270	Sample code for a simple emulator based on using the Session Level Interface (SLI) API. Sample code is provided for Win32 applications. This sample is located in the LUA\SLI3270 folder on the CD.

These samples show how to build a simple 3270 emulator using either the RUI or SLI API. This emulator does not interpret the data it receives from the host, but does demonstrate how to use the APIs to establish a session with the host.

In This Section

Building the LUA Samples

Specifying a File Name for Table G for Code Conversion

Code Samples Using the RUI API

Code Samples Using the SLI API

Reference

LUA Programmer's Reference.

Related Sections

LUA Programmer's Guide.

See Also

Other Resources

Network Integration Samples

Building the LUA Samples

The logical unit application (LUA) samples are designed to be built using Microsoft Visual C++ 6.0 or later using the command-line compiler, or using Microsoft Visual Studio .NET 2003 or later.

To build the LUA samples, set the environment variables listed in the following table.

Variable	Description
ISVLIBS	The directory containing the Host Integration Server 2009 LIB files for Microsoft Windows Server 2003 or Windows 2000 Server.
ISVINCS	The directory containing the Host Integration Server 2009 header files.
SAMPLEROOT	The root directory where the sample code provided as part of the software development kit (SDK) has been installed on a local hard disk.

For example, if you installed the Host Integration Server SDK directory to the default location (C:\Program Files\Microsoft Host Integration Server\SDK), use the following lines to set the variables (assumes Intel binaries are being produced for Windows Server 2003 or Windows 2000):

```
ISVLIBS=C:\Program Files\Microsoft Host Integration Server \SDK\LIB
ISVINCS=C:\Program Files\Microsoft Host Integration Server \SDK\Include
SAMPLEROOT=C:\Program Files\Microsoft Host Integration Server \SDK\Samples\SNA
```

To build a specific sample (the Win32 version of RUI3270, for example) using Visual Studio, open the appropriate Visual C++ project file (SNA\RUI3270\Win32\NRUI3270.vcproj, for example) from the **File** menu. Select a configuration and build the sample from the **Build** menu. Each project file has two configurations, one for a DEBUG build and one for a RETAIL build.

See Also
Other Resources
LUA Samples

Specifying a File Name for Table G for Code Conversion

The sample emulators use a user-defined table referred to as Table G in Host Integration Server 2009 for converting between ASCII and EBCDIC characters. The sample applications require that the file name of this table be specified. Use the COMTBLG registry or create a COMTBLG environment variable to specify the file name. The registry entry is described in the sections on Host Integration Server Client Binary Setup.

A sample Table G file, COMTBLG.DAT, is installed with Host Integration Server in the SYSTEM subdirectory below the root directory where the product is installed. The default location where Host Integration Server is installed is the following:

C:\Program Files\Microsoft Host Integration Server

To use this COMTBLG.DAT file, copy it to the client computers where the sample programs will be run, and specify the file name using the COMTBLG environment variable or the registry entry.

See Also

Other Resources

LUA Samples

Code Samples Using the RUI API

The following files located under the LUA\RUI3270 folder can be used to build simple 3270 emulators using the Request Unit Interface (RUI) API. Study the comments in the .C files for information about how to run the emulators.

File name	API used	Type of file	Target operating system
RUI3270.C	RUI	Source code for RUI3270.EXE	Microsoft® Windows Server™ 2003 or Windows® 2000 Server
Win32\NRUI3270.MAK	RUI	Makefile	Windows Server 2003 or Windows 2000
Win32\RUIINC.C	RUI	Source code that #includes RUI3270.c to build a Win32 version of WRUI3270.EXE	Windows Server 2003 or Windows 2000
Win32\WRUI3270.RC	RUI	Resource definition file	Windows Server 2003 or Windows 2000

See Also

Other Resources

LUA Samples

Code Samples Using the SLI API

The following files located under the LUA\SLI3270 folder can be used to build simple 3270 emulators using the Session Level Interface (SLI) API. Study the comments in the .C files for information about how to run the emulators.

File name	API used	Type of file	Target operating system
SLI3270.C	SLI	Source code for SLI3270.EXE	Microsoft® Windows Server™ 2003 or Windows® 2000 Server
Win32\NSLI3270.MAK	SLI	Makefile	Windows Server 2003 or Windows 2000
Win32\SLIINC.C	SLI	Source code that #includes SLI3270.c to build a Win32 version of SLI3270.EXE	Windows Server 2003 or Windows 2000
Win32\WSLI3270.RC	SLI	Resource definition file	Windows Server 2003 or Windows 2000

See Also

Other Resources

LUA Samples

SNA Print Server Data Filter Samples

The PrintServer sample illustrates features of the SNA Print Server Data Filter API. These features can be used to extend the capabilities of the Host Print Service in Host Integration Server 2009. The sample code illustrates how to write a print data filter dynamic-link library (DLL) that will be called by Host Print Service when a print job is initiated, when data is sent to the printer, and when the print job is completed.

Location in the SDK

<installation directory>\Program Files\<version>\SDK\Samples\NetworkIntegration\PrinServer

File Inventory

File(s)	Description
In the /PrintServer/PrintDefFile directory Hplj2.pdf	The sample file to print
In the /PrintServer/PrnFltr directory Makefile NTFilter.c NTFilter.Def NtFilter.h NTFilter.Mdp NTFilter.rc Prntfltr.sln Prntfltr.vcproj	The sample application an related files, The sample program written in C that illustrates use of the SNA Print Server Data Filter API. Also includes an include file with function defines, a resource file, a command-line makefile, and a project file.

How to Use the Sample

The PrnFltr sample is designed to be built using Visual Studio .NET 2003 or later.

To build the PrnFltr sample using the command-line compiler

1. Open a command prompt window
2. Run VSVARS32.bat from the Visual Studio bin directory

   by default, the location is C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools
3. Navigate to \PrintServer\PrnFlter subdirectory, and invoke NMAKE

To build the PrnFltr sample using Visual Studio

1. open the appropriate Visual C++ project file (PrintServer\PrnFltr\ntfilter.vcproj) from the **File** menu.
2. Select a configuration and build the sample from the **Build** menu.

   Each project file has two configurations, one for a DEBUG build and one for a RETAIL build.

See Also

Other Resources

Network Integration Samples
SNA Print Server Data Filter Programmer's Reference
SNA Print Server Data Filter Programmer's Guide

Session Integrator Samples

This section contains descriptions of the sample applications that describe the basic capabilities of the Session Integrator technology,

In This Section

3270NET Sample

LU0NET Sample

Reference

Microsoft.HostIntegration.SNA.Session

Session Integrator Programmer's Reference

Related Sections

Session Integrator Programmer's Guide

See Also

Other Resources

Network Integration Samples

3270NET Sample

The 3270Net sample shows how to create a 3270 session using a .NET application.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\NetworkIntegration\SessionIntegrator\3270Net

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /3270Net folder 3270LogonScriptReadme.txt 3270TutorialSolution.sln	Contains the solution file. Also contains the readme, which has detailed instructions for setting up and deploying the application
In the /3270Net/LogonScriptClient folder Form1.cs Form1.Designer.cs Form1.resx LogonClient.csproj LogonClient.sln Program.cs	Contains the primary files for the application.
In the /3270Net/LogonScriptClient/Properties folder AssemblyInfo.cs Resources.Designer.cs Resources.resx Settings.Designer.cs Settings.settings	Contains the assembly and resources for the sample.

See Also

Reference

Microsoft.HostIntegration.SNA.Session

Other Resources

Session Integrator Programmer's Guide

Session Integrator Samples

LU0NET Sample

The LU0NET sample shows how to create a LU0 session using a .NET application.

Location in SDK

<installation directory>:\Program Files\Microsoft Host Integration Server\SDK\Samples\NetworkIntegration\SessionIntegrator\LU0

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
In the /LU0 folder LU0LogonScriptReadme.txt LU0TutorialSolution.sln	Contains the solution file. Also contains the readme, which has detailed instructions for setting up and deploying the application
In the /LU0/LogonScriptClient folder Form1.cs Form1.Designer.cs Form1.resx LogonClient.csproj LogonClient.sln Program.cs	Contains the primary files for the application.
In the /LU0/LogonScriptClient/Properties folder AssemblyInfo.cs Resources.Designer.cs Resources.resx Settings.Designer.cs Settings.settings	Contains the assembly and resources for the sample.

See Also

Reference

Microsoft.HostIntegration.SNA.Session

Other Resources

Session Integrator Programmer's Guide

Session Integrator Samples

Single Sign-On Samples

This section of the Host Integration Server 2009 Developer's Guide describes the sample applications that implement Single Sign-On programming technology.

In This Section

Loopback Adapter Sample

Reference

Microsoft.EnterpriseSingleSignOn.Interop

Loopback Adapter Sample

The Loopback Adapter sample is a password sync adapter for Enterprise Single-Sign on (ENTSSO) implemented as a windows NT service written in C#. The sample demonstrates how to use the SSOPSHelper COM component and the ISSOPSWrapper interface to develop a password sync adapter in managed code.

Location in the SDK

<drive>\Program Files\Common Files\Enterprise Single Sign-On\SDK\Samples\LoopbackAdapter

File Inventory

The following table shows the files in the sample and describes their purpose.

File(s)	Description
AssemblyInfo.cs	Contains the assembly information for the project.
Createadapter_LoopbackAdapter.xml	Used with the ssops command-line tool to create the prototype adapter.
LoopbackAdapter.cs LoopbackAdapter.csproj LoopbackAdapter.Designer.cs LoopbackAdapter.resx	Contains the sample code for crating the Loopback adapter.
LoopbackAdapter.sln	Organizes all the projects, project items, and solution items into the Loopback Solution by providing the environment with references to their locations on disk. Use Visual Studio to view this file, or double-click the file to start Visual Studio 2005 and displays the solution in Solution Explorer.
LoopbackAdapter.suo	Records all of the options associated with the Loopback Adapter Solution so that each time you open it, it includes customizations that you have made.
ProjectInstaller.cs ProjectInstaller.Designer.cs ProjectInstaller.resx	Contains the sample code for the project installer.
Properties_LoopbackAdapter.xml	Property definitions for the Loopback Adapter.
Program.cs	Contains the basic framework of the application.
README.txt	Shows the steps to build and run the application.

See Also

Other Resources

Single Sign-On Samples